Synchronous vs Asynchronous Digital Circuits as an Analogy to Organizational Dysfunction and Applied to DevOps Practices

WTF is this talk about?

A Tiny Bit of Theory About Digital Electronics - Synchronous vs Asynchronous

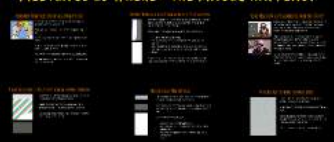
Famous Asynchronous Digital Logic Devices That You May Be Familiar With

Why Asynchronous is a Hot Topic in Digital Logic Design Today

How and Why are Organizations Becoming More Like Synchronous Digital Circuits?

Are You Saying That All the SAFe "Flow Model" Stuff is All Wrong?

Where did organizations get the idea that meetings is where "the magic happens?"



OK. Now, as an Organization, Let's Improve and Reduce Meetings. What Would That Look Like?

On Virtualization and Containerization and Why They Can Help

Closing Thoughts and Takeaways

WTF is this talk about?

About Me

Programming since High School in 1968 (some day, I'll figure it out) (Code Retro IBM 1130)

Working full-time since 1975 (while still a junior in college)

Have had the pleasure of working at all levels (individual contributor to executive), both commercial and scientific

Have been an Agile Coach since 2003 (so few of us back then that it seemed that we all knew one another)

To be successful back then, you had to be both a process and Extreme Programming Coach

Throughout my life, two things have fascinated me
- Digital electronics
- Software Development

"The Wastes are Water", and Other Advice from the Older Agile Fish.

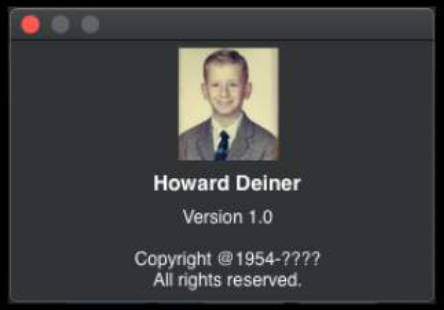
A few years ago, I wrote an article for SolutionsIQ based on something that I heard in a talk by Neal Ford, who was quoting from a commencement address given to the 2005 class of Kenyon College by David Wallace. (That's what full disclosure looks like)

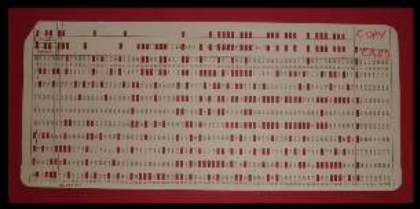
Anyway, being the Agilist and Lean Practioner that I am, it struck me deeply.

I hope to be able to give you just a bit of a view into what I feel is a terrible affliction that we suffer under: we live in a world filled with clock signals.

Bear with me a moment while I give you the background to see what I'm talking about.

About Me





Programming since High School in 1968 (some day, I'll figure it out) [cool Retro IBM 1130]

Working full-time since 1975 (while still a junior in college)

Have had the pleasure of working at all levels (individual contributor to executive), both commercial and scientific

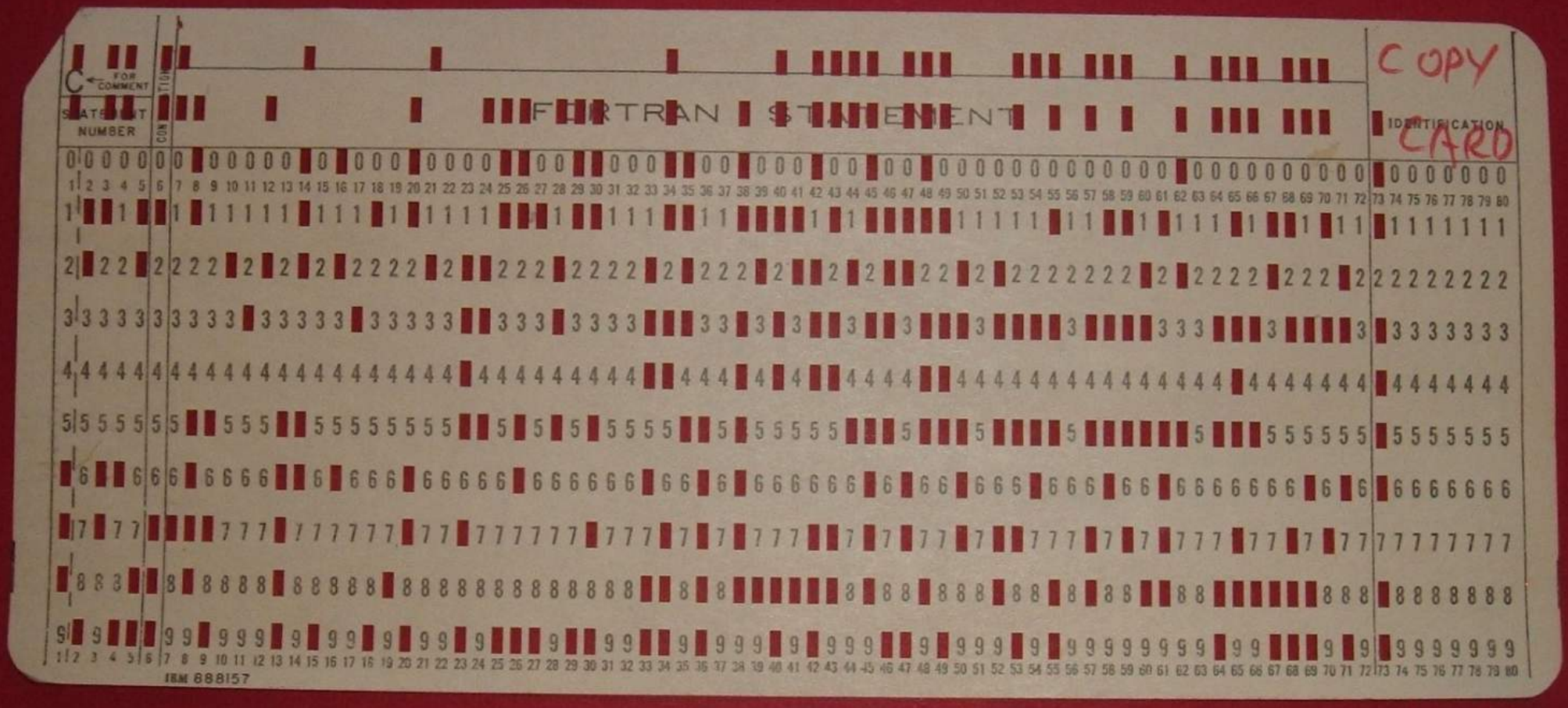
Have been an Agile Coach since 2003 (so few of us back then that it seemed that we all knew one another)

To be successful back then, you had to be both a process and eXtreme Programming Coach

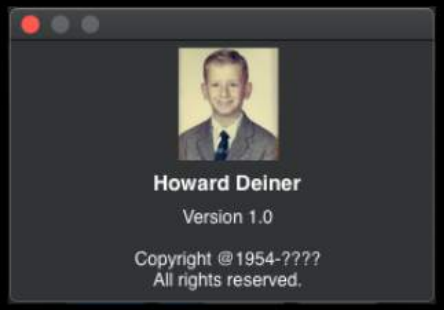
Throughout my life, two things have facinated me

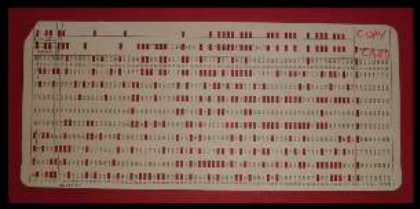
- Digital electronics
- Software Development


Historisches
aus dem
Computer
IBM
IBM
IBM
CALCOMP


C FOR COMMENT
STATEMENT NUMBER
CONTINUATION
FORTRAN STATEMENT
IDENTIFICATION
COPY
CARD
IBM 888157

About Me





Programming since High School in 1968 (some day, I'll figure it out) [cool Retro IBM 1130]

Working full-time since 1975 (while still a junior in college)

Have had the pleasure of working at all levels (individual contributor to executive), both commercial and scientific

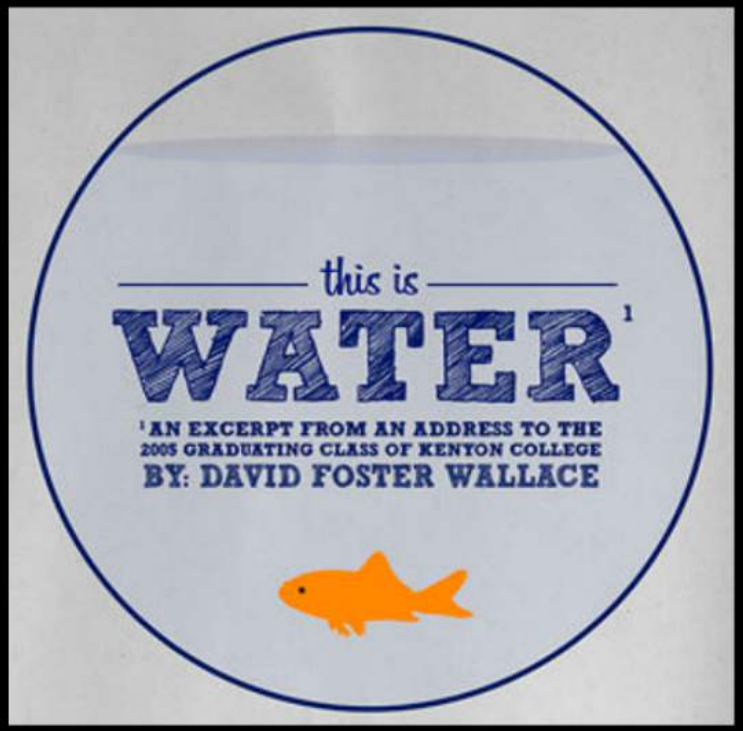
Have been an Agile Coach since 2003 (so few of us back then that it seemed that we all knew one another)

To be successful back then, you had to be both a process and eXtreme Programming Coach

Throughout my life, two things have facinated me

- Digital electronics
- Software Development

"The Wastes are Water", and Other Advice from the Older Agile Fish.



A few years ago, I wrote an article for SolutionsIQ based on something that I heard in a talk by Neal Ford, who was quoting from a commencement address given to the 2005 class of Kenyon College by David Wallace. [that's what full disclosure looks like]

Anyway, being the Agilist and Lean Practioner that I am, it struck me deeply.

I hope to be able to give you just a bit of a view into what I feel is a terrible affliction that we suffer under: we live in a world filled with clock signals.

Bear with me a moment while I give you the background to see what I'm talking about.

WTF is this talk about?

About Me

Programming since High School in 1968 (some day, I'll figure it out) (Code Retro IBM 1130)

Working full-time since 1975 (while still a junior in college)

Have had the pleasure of working at all levels (individual contributor to executive), both commercial and scientific

Have been an Agile Coach since 2003 (so few of us back then that it seemed that we all knew one another)

To be successful back then, you had to be both a process and Extreme Programming Coach

Throughout my life, two things have fascinated me
- Digital electronics
- Software Development

"The Wastes are Water", and Other Advice from the Older Agile Fish.

A few years ago, I wrote an article for SolutionsIQ based on something that I heard in a talk by Neal Ford, who was quoting from a commencement address given to the 2005 class of Kenyon College by David Wallace. (That's what full disclosure looks like)

Anyway, being the Agilist and Lean Practioner that I am, it struck me deeply.

I hope to be able to give you just a bit of a view into what I feel is a terrible affliction that we suffer under: we live in a world filled with clock signals.

Bear with me a moment while I give you the background to see what I'm talking about.

WTF is this talk about?

A Tiny Bit of Theory About Digital Electronics - Synchronous vs Asynchronous

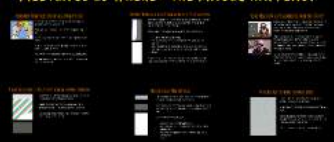
Famous Asynchronous Digital Logic Devices That You May Be Familiar With

Why Asynchronous is a Hot Topic in Digital Logic Design Today

How and Why are Organizations Becoming More Like Synchronous Digital Circuits?

Are You Saying That All the SAFe "Flow Model" Stuff is All Wrong?

Where did organizations get the idea that meetings is where "the magic happens?"



OK. Now, as an Organization, Let's Improve and Reduce Meetings. What Would That Look Like?

On Virtualization and Containerization and Why They Can Help

Closing Thoughts and Takeaways

A Tiny Bit of Theory About Digital Electronics - Synchronous vs Asynchronous

Let's Start With Synchronous Circuits

In synchronous circuits, an electronic oscillator generates a repetitive series of equally spaced pulses called the clock signal.

The clock signal is applied to all the elements in the circuit.

The outputs of the circuit only change when triggered by the edge of the clock pulse, so changes to the logic signals throughout the circuit all begin at the same time, at regular intervals synchronized by the clock.

The output of all elements in a circuit is called the state of the circuit.

The state of a synchronous circuit changes only on the clock pulse.

More About Synchronous Circuits

The changes in signal require a certain amount of time to propagate through the combinational logic gates of the circuit.

This is called propagation delay.

The period of the clock signal is made long enough so the output of all the logic gates have time to settle to stable values before the next clock pulse.

As long as this condition is met, synchronous circuits will operate stably, so they are easy to design.

So, What's Wrong With Synchronous Circuits?

A disadvantage of synchronous circuits is that they can be slow.

The maximum possible clock rate is determined by the logic path with the longest propagation delay.

So logic paths that complete their operations quickly are idle most of the time.

Moving Along, Let's Consider Asynchronous Circuits

In asynchronous circuits, there is no clock signal, and the state of the circuit changes as soon as the inputs change.

Since asynchronous circuits don't have to wait for a clock pulse to begin processing inputs, they can be faster than synchronous circuits, and their speed is theoretically limited only by the propagation delays of the logic gates themselves.

Asynchronous Circuit Benefits

- Higher performance, because completion is based on the actual input values, rather than worst-case completion.
- Lower power consumption because no transistor ever transitions unless it is performing useful computation.
- Freedom from the ever-worsening difficulties of distributing a high-fan-out, timing-sensitive clock signal.
- Better modularity and composability.
- Circuit's speed adapts to changing temperature and voltage conditions rather than being locked at the speed mandated by worst-case assumptions.

Even More Asynchronous Circuit Benefits

Immunity to transistor-to-transistor variability in the manufacturing process, which is one of the most serious problems facing the semiconductor industry as dies shrink.

Less severe electromagnetic interference (EMI). Synchronous circuits create a great deal of EMI in the frequency band of their clock frequency and its harmonics. Asynchronous circuits generate EMI patterns which are much more evenly spread across the spectrum.

Less stress on the power distribution network. Synchronous circuits tend to draw a large amount of current right at the clock edge and shortly thereafter. The number of nodes switching (and thence, amount of current drawn) drops off rapidly after the clock edge, reaching zero just before the next clock edge. In an asynchronous circuit, the switching times of the nodes are not correlated in this manner, so the current draw tends to be more uniform and less bursty.

Let's Start With Synchronous Circuits

In synchronous circuits, an electronic oscillator generates a repetitive series of equally spaced pulses called the clock signal.

The clock signal is applied to all the elements in the circuit.

The outputs of the circuit only change when triggered by the edge of the clock pulse, so changes to the logic signals throughout the circuit all begin at the same time, at regular intervals synchronized by the clock.

The output of all elements in a circuit is called the state of the circuit.

The state of a synchronous circuit changes only on the clock pulse.

More About Synchronous Circuits

The changes in signal require a certain amount of time to propagate through the combinational logic gates of the circuit.

This is called propagation delay.

The period of the clock signal is made long enough so the output of all the logic gates have time to settle to stable values before the next clock pulse.

As long as this condition is met, synchronous circuits will operate stably, so they are easy to design.

So, What's Wrong With Synchronous Circuits?

A disadvantage of synchronous circuits is that they can be slow.

The maximum possible clock rate is determined by the logic path with the longest propagation delay.

So logic paths that complete their operations quickly are idle most of the time.

Moving Along, Let's Consider Asynchronous Circuits

In asynchronous circuits, there is no clock signal, and the state of the circuit changes as soon as the inputs change.

Since asynchronous circuits don't have to wait for a clock pulse to begin processing inputs, they can be faster than synchronous circuits, and their speed is theoretically limited only by the propagation delays of the logic gates themselves.

Asynchronous Circuit Benefits

Higher performance, because completion is based on the actual input values, rather than worst-case completion.

Lower power consumption because no transistor ever transitions unless it is performing useful computation.

Freedom from the ever-worsening difficulties of distributing a high-fan-out, timing-sensitive clock signal.

Better modularity and composability.

Circuit speed adapts to changing temperature and voltage conditions rather than being locked at the speed mandated by worst-case assumptions.

Even More Asynchronous Circuit Benefits

Immunity to transistor-to-transistor variability in the manufacturing process, which is one of the most serious problems facing the semiconductor industry as dies shrink.

Less severe electromagnetic interference (EMI). Synchronous circuits create a great deal of EMI in the frequency band of their clock frequency and its harmonics; asynchronous circuits generate EMI patterns which are much more evenly spread across the spectrum.

Less stress on the power distribution network. Synchronous circuits tend to draw a large amount of current right at the clock edge and shortly thereafter. The number of nodes switching (and thence, amount of current drawn) drops off rapidly after the clock edge, reaching zero just before the next clock edge. In an asynchronous circuit, the switching times of the nodes are not correlated in this manner, so the current draw tends to be more uniform and less bursty.

A Tiny Bit of Theory About Digital Electronics - Synchronous vs Asynchronous

Let's Start With Synchronous Circuits

In synchronous circuits, an electronic oscillator generates a repetitive series of equally spaced pulses called the clock signal.

The clock signal is applied to all the elements in the circuit.

The outputs of the circuit only change when triggered by the edge of the clock pulse, so changes to the logic signals throughout the circuit all begin at the same time, at regular intervals synchronized by the clock.

The output of all elements in a circuit is called the state of the circuit.

The state of a synchronous circuit changes only on the clock pulse.

More About Synchronous Circuits

The changes in signal require a certain amount of time to propagate through the combinational logic gates of the circuit.

This is called propagation delay.

The period of the clock signal is made long enough so the output of all the logic gates have time to settle to stable values before the next clock pulse.

As long as this condition is met, synchronous circuits will operate stably, so they are easy to design.

So, What's Wrong With Synchronous Circuits?

A disadvantage of synchronous circuits is that they can be slow.

The maximum possible clock rate is determined by the logic path with the longest propagation delay.

So logic paths that complete their operations quickly are idle most of the time.

Moving Along, Let's Consider Asynchronous Circuits

In asynchronous circuits, there is no clock signal, and the state of the circuit changes as soon as the inputs change.

Since asynchronous circuits don't have to wait for a clock pulse to begin processing inputs, they can be faster than synchronous circuits, and their speed is theoretically limited only by the propagation delays of the logic gates themselves.

Asynchronous Circuit Benefits

- Higher performance, because completion is based on the actual input values, rather than worst-case completion.
- Lower power consumption because no transistor ever transitions unless it is performing useful computation.
- Freedom from the ever-worsening difficulties of distributing a high-fan-out, timing-sensitive clock signal.
- Better modularity and composability.
- Circuit's speed adapts to changing temperature and voltage conditions rather than being locked at the speed mandated by worst-case assumptions.

Even More Asynchronous Circuit Benefits

Immunity to transistor-to-transistor variability in the manufacturing process, which is one of the most serious problems facing the semiconductor industry as dies shrink.

Less severe electromagnetic interference (EMI). Synchronous circuits create a great deal of EMI in the frequency band of their clock frequency and its harmonics. Asynchronous circuits generate EMI patterns which are much more evenly spread across the spectrum.

Less stress on the power distribution network. Synchronous circuits tend to draw a large amount of current right at the clock edge and shortly thereafter. The number of nodes switching (and thence, amount of current drawn) drops off rapidly after the clock edge, reaching zero just before the next clock edge. In an asynchronous circuit, the switching times of the nodes are not correlated in this manner, so the current draw tends to be more uniform and less bursty.

WTF is this talk about?

A Tiny Bit of Theory About Digital Electronics - Synchronous vs Asynchronous

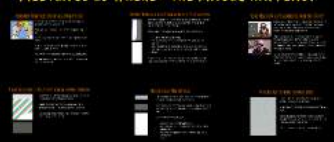
Famous Asynchronous Digital Logic Devices That You May Be Familiar With

Why Asynchronous is a Hot Topic in Digital Logic Design Today

How and Why are Organizations Becoming More Like Synchronous Digital Circuits?

Are You Saying That All the SAFe "Flow Model" Stuff is All Wrong?

Where did organizations get the idea that meetings is where "the magic happens?"



OK. Now, as an Organization, Let's Improve and Reduce Meetings. What Would That Look Like?

On Virtualization and Containerization and Why They Can Help

Closing Thoughts and Takeaways

Famous Asynchronous Digital Logic Devices That You May Be Familiar With

The ILLIAC I - 1952

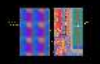
Asynchronous Microprocessors, East and West

The Philips Semiconductors asynchronous 80C51 microcontroller

Intel/Fulcrum Microsystems: Ethernet Switch Chips

Achronix: high-performance FPGAs

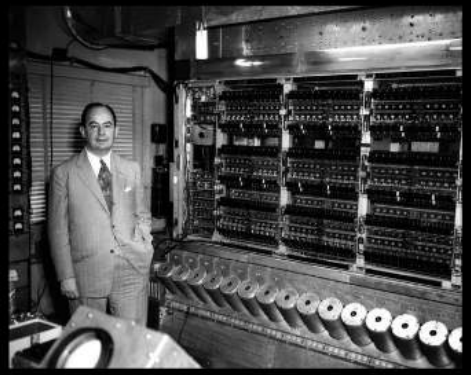
The IBM TrueNorth Neuromorphic Computer



Oh, BTW, Computers and Brains Aren't Too Much Alike

The ILS - 1952

Built from 1945 to 1951 at Institute for Advanced Study (IAS) under the direction of John von Neumann, and proved the Von Neumann architecture (still in use today by the vast majority of CPUs).



Von Neumann architecture generally means

- A processing unit that contains an arithmetic logic unit and processor registers
- A control unit that contains an instruction register and program counter
- Memory that stores data and instructions
- External mass storage
- Input and output mechanisms

The ILS was a binary computer with a 40-bit word, storing two 20-bit instructions in each word

Memory was 1,024 words (5.1 kilobytes)

Negative numbers were represented in "two's complement" format

Had two general-purpose registers available: the Accumulator (AC) and Multiplier/Quotient (MQ)

Used 1,700 vacuum tubes

Memory was originally designed for about 2,300 RCA Selectron vacuum tubes.

Problems with the development of these complex tubes forced the switch to Williams tubes.

A grid of dots was displayed on a cathode ray tube, creating a small charge of static electricity over each dot, which a thin sheet of metal just in front of the display would read.

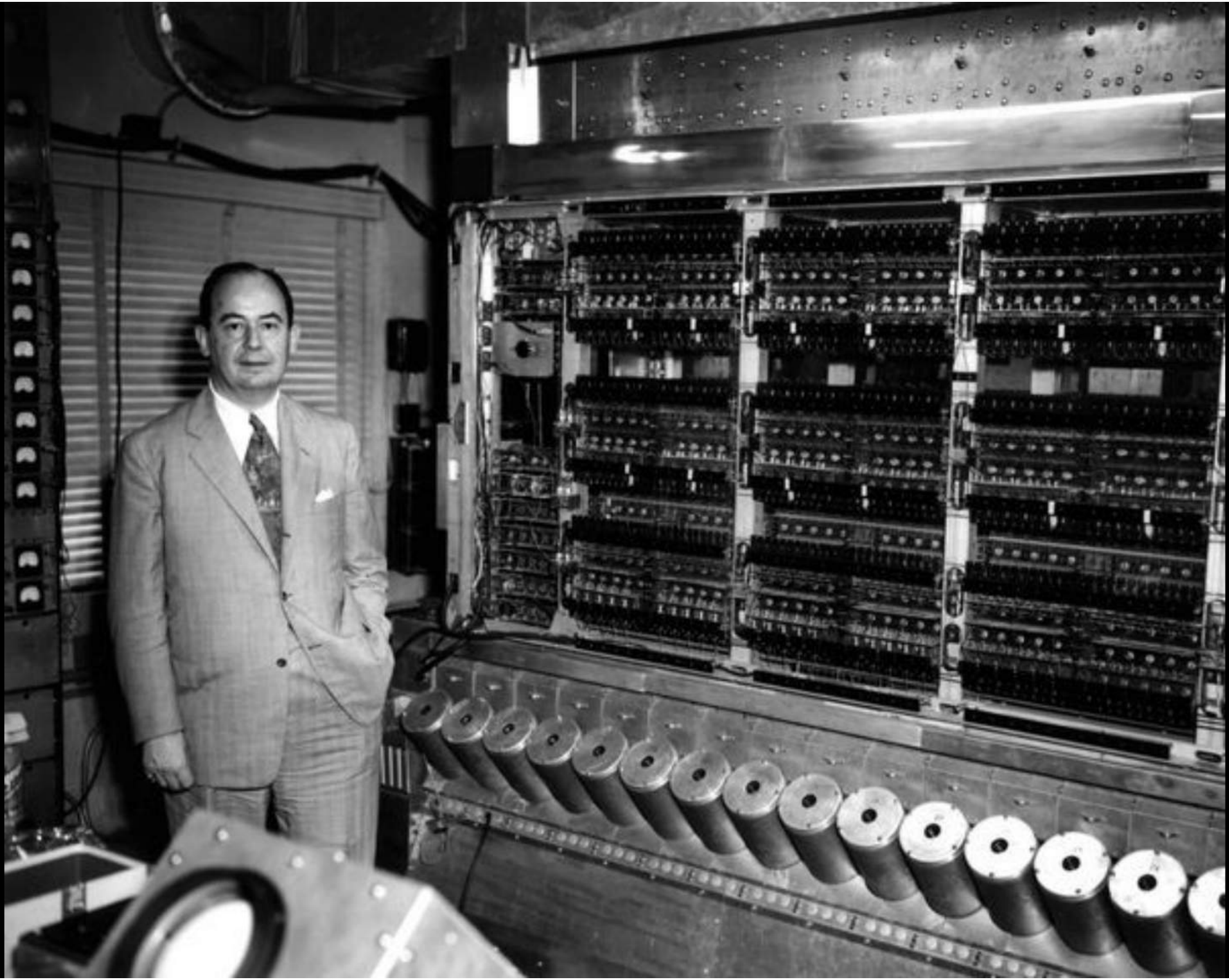
While there was persistence on the surface of the tube for a fraction of a second, memory needed to be rewritten in a similar operation to the memory refresh cycles of DRAM in modern systems.

Measured 3m by 0.3m by 2.2m

Weighed 450 kg

Since it was an asynchronous machine, instruction times varied: addition time was 62 microseconds and the multiplication time was 713 microseconds.

Built from 1945 to 1951 at Institute for Advanced Study (IAS) under the direction of John von Neumann, and proved the Von Neumann architecture (still in use today by the vast majority of CPUs).



Von Neumann architecture generally means

- A processing unit that contains an arithmetic logic unit and processor registers
- A control unit that contains an instruction register and program counter
- Memory that stores data and instructions
- External mass storage
- Input and output mechanisms

The ILS was A binary computer with a 40-bit word, storing two 20-bit instructions in each word

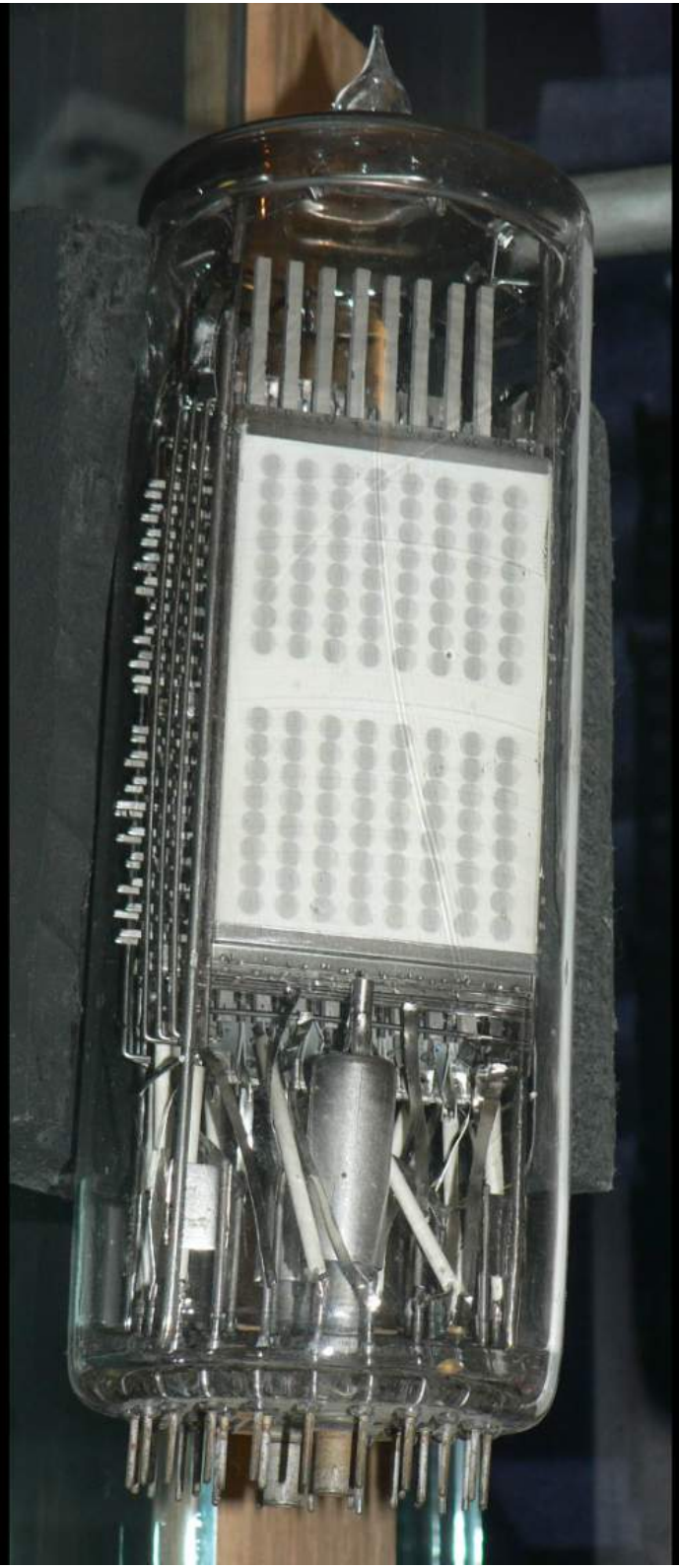
memory was 1,024 words (5.1 kilobytes)

negative numbers were represented in "two's complement" format

had two general-purpose registers available: the Accumulator (AC) and Multiplier/ Quotient (MQ)

used 1,700 vacuum tubes

memory was originally designed for about 2,300 RCA Selectron vacuum tubes.



Problems with the development of these complex tubes forced the switch to Williams tubes.

A grid of dots was displayed on a cathode ray tube, creating a small charge of static electricity over each dot, which a thin sheet of metal just in front of the display would read.

While there was persistence on the surface of the tube for a fraction of a second, memory needed to be rewritten in a similar operation to the memory refresh cycles of DRAM in modern systems.


IBM

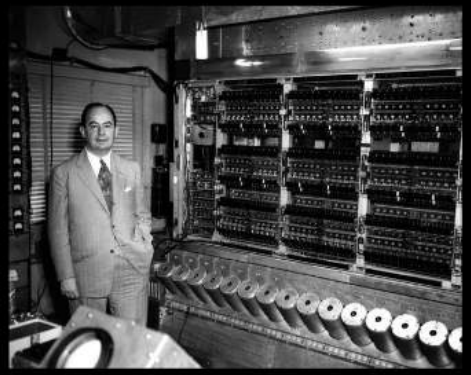
measured 3m by 0.8m by 3.2m

weighed 450 kg

since it was an asynchronous machine, instruction times varied: addition time was 62 microseconds and the multiplication time was 713 microseconds.

The ILS - 1952

Built from 1945 to 1951 at Institute for Advanced Study (IAS) under the direction of John von Neumann, and proved the Von Neumann architecture (still in use today by the vast majority of CPUs).



Von Neumann architecture generally means

- A processing unit that contains an arithmetic logic unit and processor registers
- A control unit that contains an instruction register and program counter
- Memory that stores data and instructions
- External mass storage
- Input and output mechanisms

The ILS was a binary computer with a 40-bit word, storing two 20-bit instructions in each word

Memory was 1,024 words (5.1 kilobytes)

Negative numbers were represented in "two's complement" format

Had two general-purpose registers available: the Accumulator (AC) and Multiplier/Quotient (MQ)

Used 1,700 vacuum tubes

Memory was originally designed for about 2,300 RCA Selectron vacuum tubes.

Problems with the development of these complex tubes forced the switch to Williams tubes.

A grid of dots was displayed on a cathode ray tube, creating a small charge of static electricity over each dot, which a thin sheet of metal just in front of the display would read.

While there was persistence on the surface of the tube for a fraction of a second, memory needed to be rewritten in a similar operation to the memory refresh cycles of DRAM in modern systems.

Measured 3m by 0.3m by 2.2m

Weighed 450 kg

Since it was an asynchronous machine, instruction times varied: addition time was 62 microseconds and the multiplication time was 713 microseconds.

Famous Asynchronous Digital Logic Devices That You May Be Familiar With

The ILLIAC I - 1952

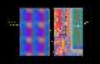
Asynchronous Microprocessors, East and West

The Philips Semiconductors asynchronous 80C51 microcontroller

Intel/Fulcrum Microsystems: Ethernet Switch Chips

Achronix: high-performance FPGAs

The IBM TrueNorth Neuromorphic Computer



Oh, BTW, Computers and Brains Aren't Too Much Alike

The ILLIAC I - 1952

First computer built and owned entirely by a US educational institution.

First computer to share an instruction set with another computer (this was the second of identical machines, the first of which was ORDVAC [US Army Aberdeen Proving Grounds], also built at the University of Illinois.)

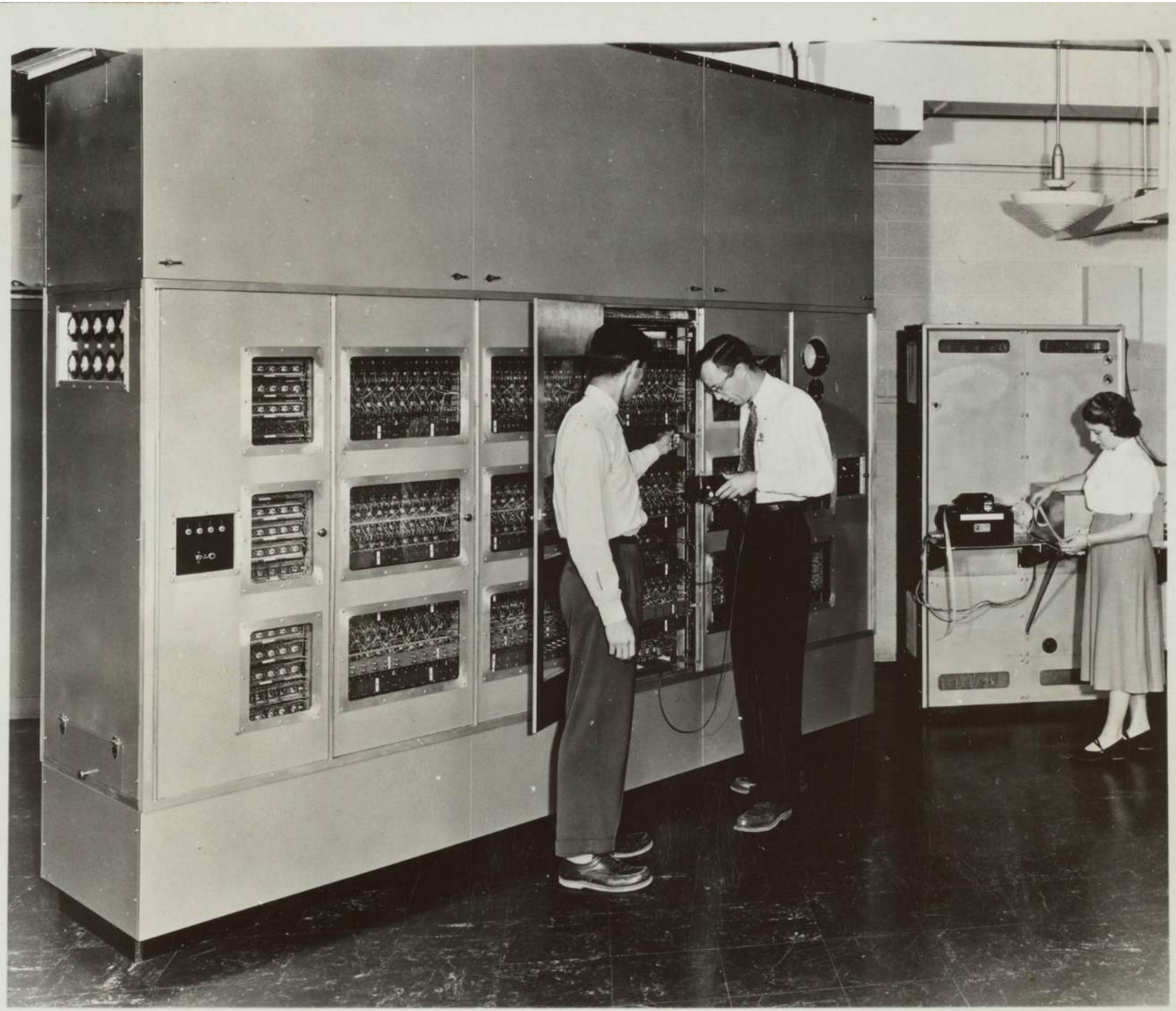
had 2,800 vacuum tubes

measured 3 m by 0.6 m by 2.6 m

weighed 2668 kg

was very powerful for its time:in 1956 it had more computing power than all of Bell Labs.

because the lifetime of the tubes within ILLIAC was about a year, the machine was shut down every day for "preventive maintenance" when older vacuum tubes would be replaced in order to increase reliability.



The ILLIAC I - 1952

First computer built and owned entirely by a US educational institution.

First computer to share an instruction set with another computer (this was the second of identical machines, the first of which was ORDVAC [US Army Aberdeen Proving Grounds], also built at the University of Illinois.)

had 2,800 vacuum tubes

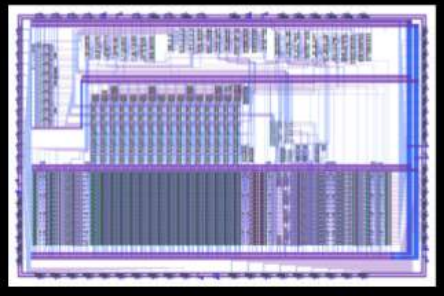
measured 3 m by 0.6 m by 2.6 m

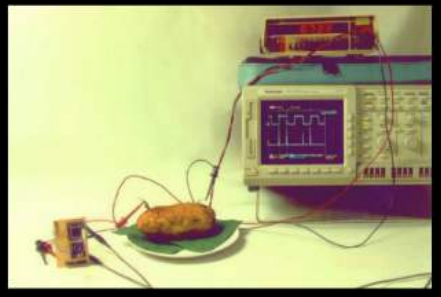
weighed 2668 kg

was very powerful for its time:in 1956 it had more computing power than all of Bell Labs.

because the lifetime of the tubes within ILLIAC was about a year, the machine was shut down every day for "preventive maintenance" when older vacuum tubes would be replaced in order to increase reliability.

Asynchronous Microprocessors, East and West

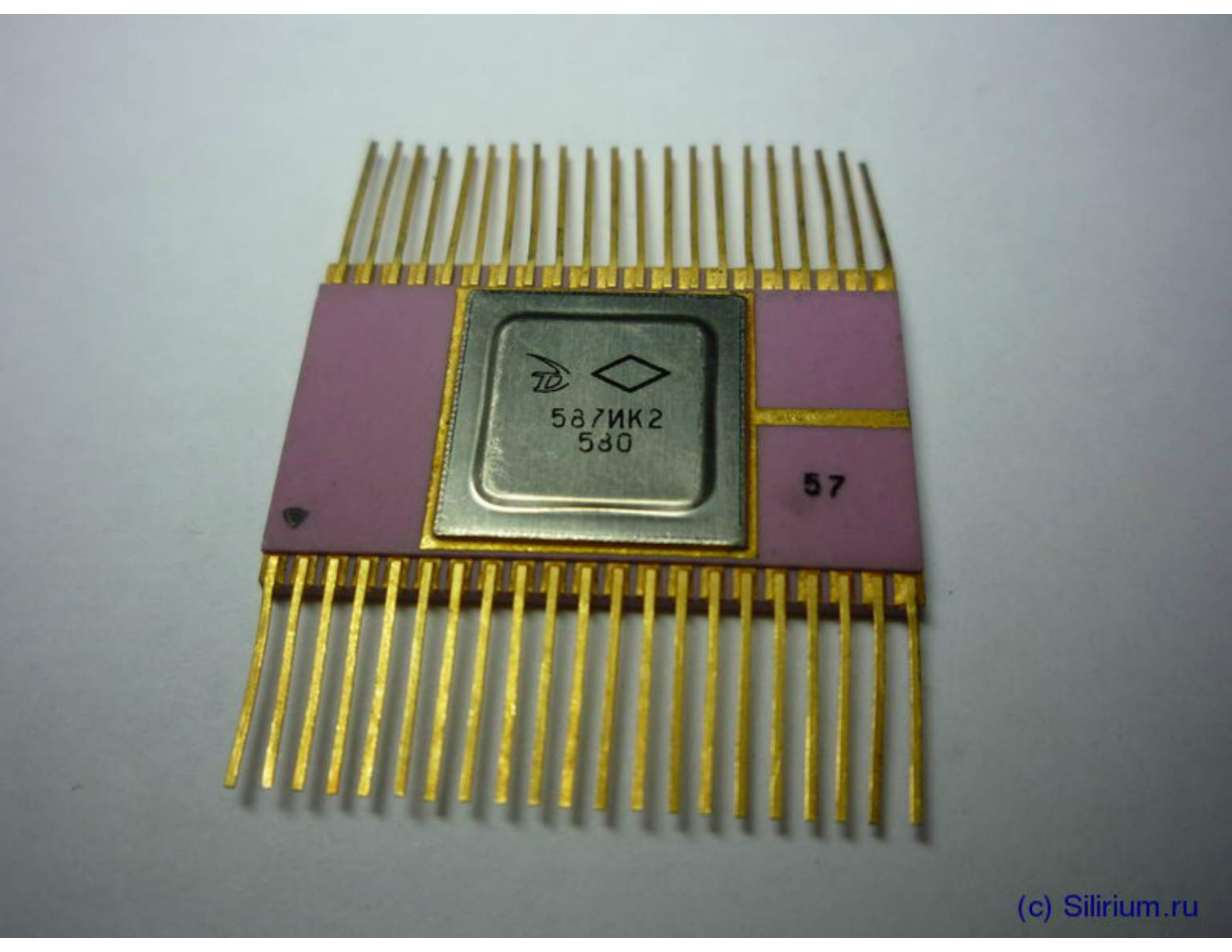




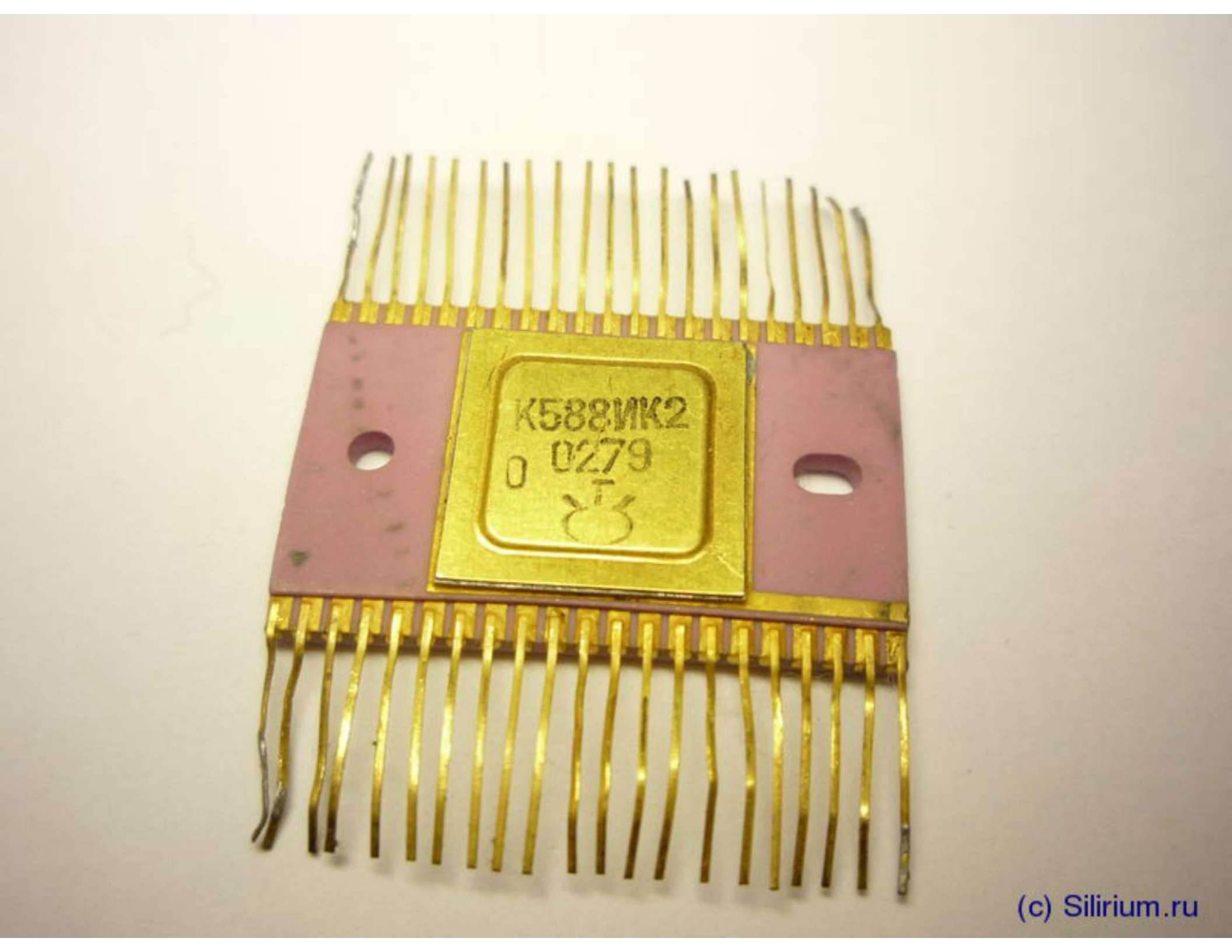
Soviet bit-slice microprocessor modules (late 1970s) produced as

- K587 (Elektronika NC microarchitecture core, with PDP-11 compatible instruction set)
- K588 (PDP-11 instruction set)
- K1883 (controller, also known as U83x in East Germany)

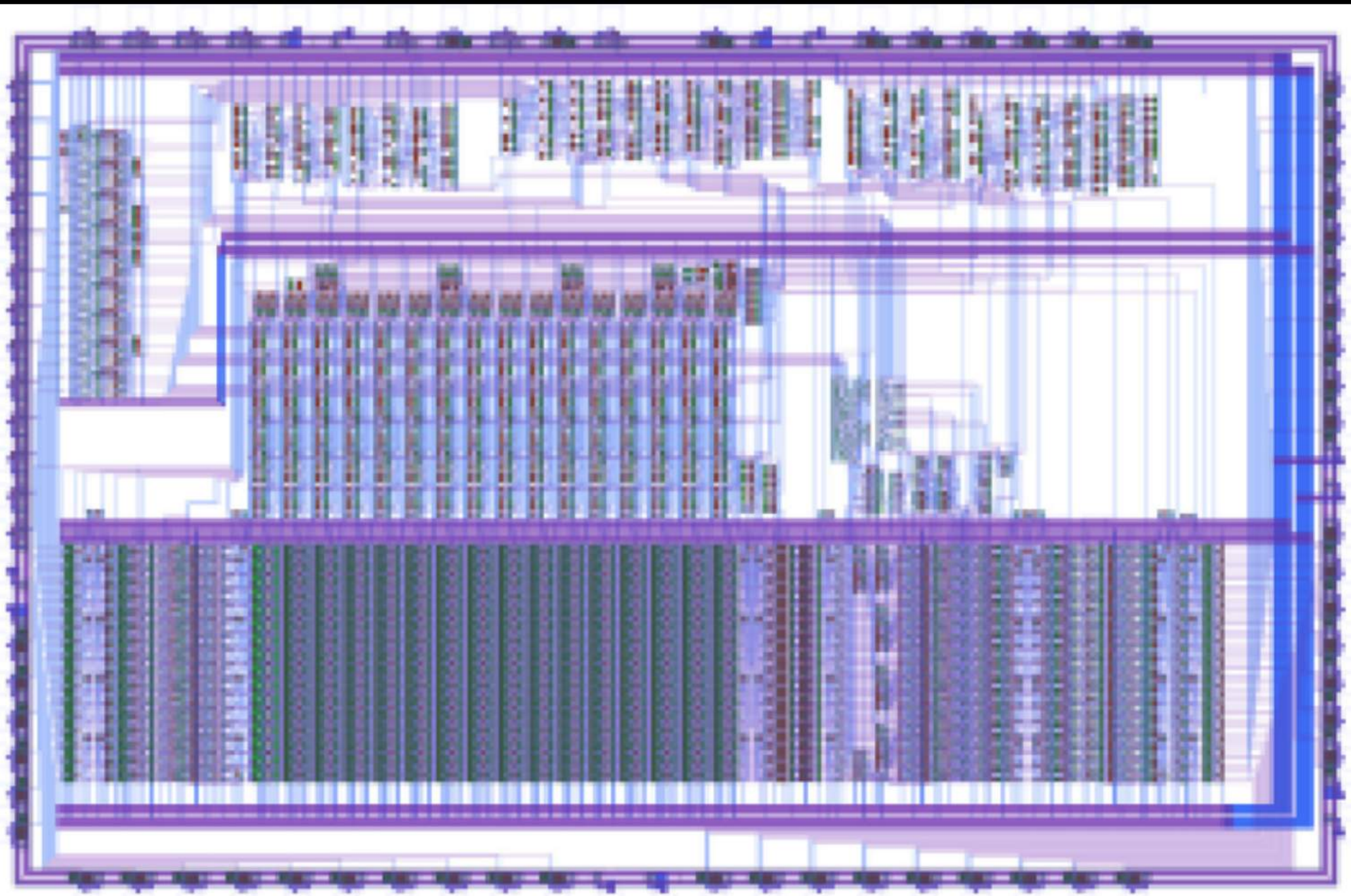
The Caltech Asynchronous Microprocessor (CAM), the world-first asynchronous microprocessor (1988).

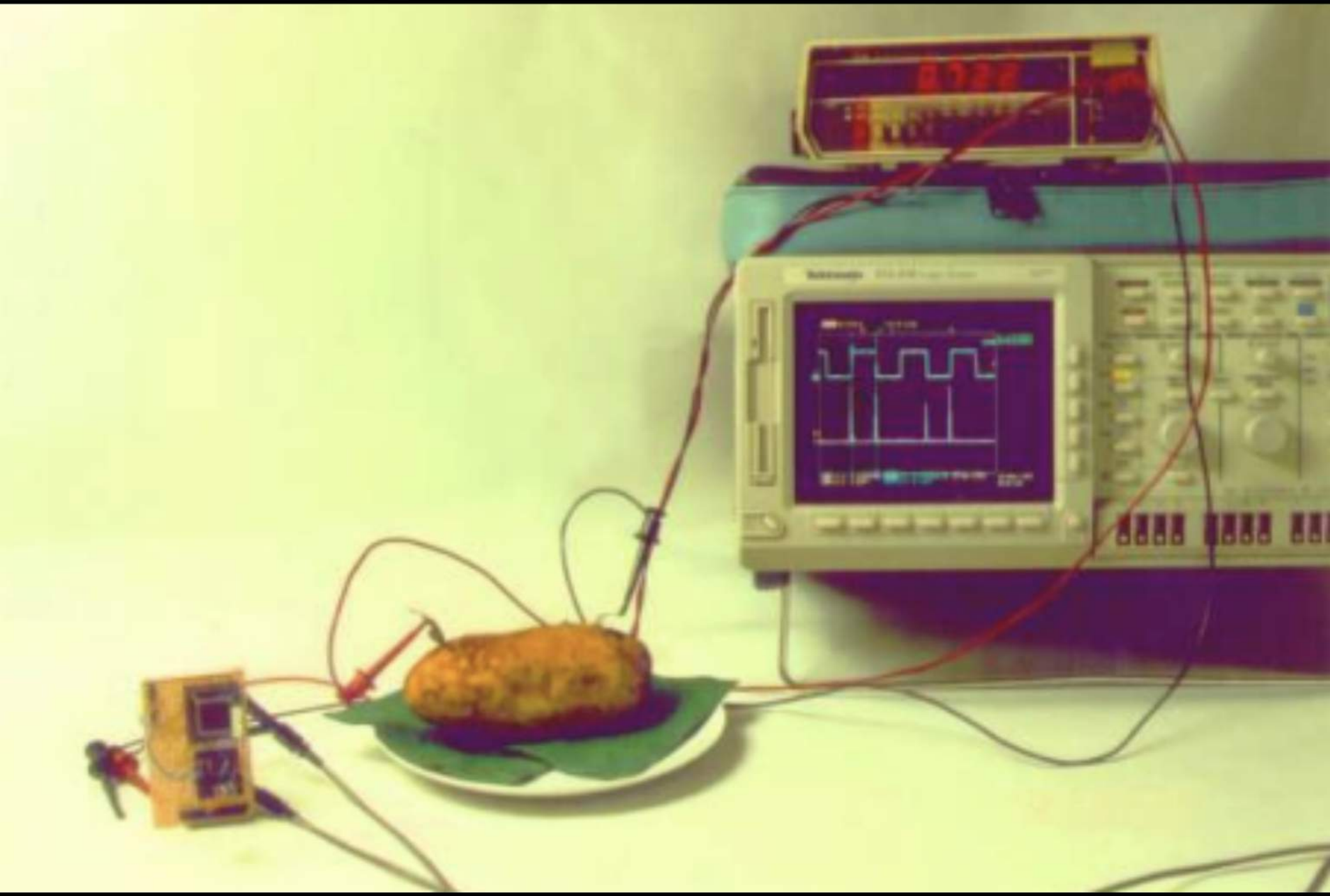


(c) Silirium.ru

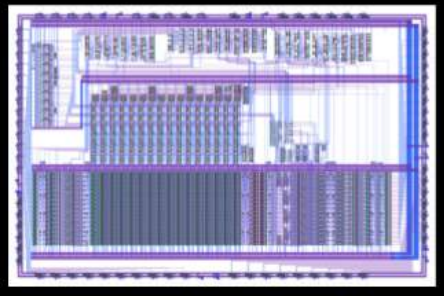


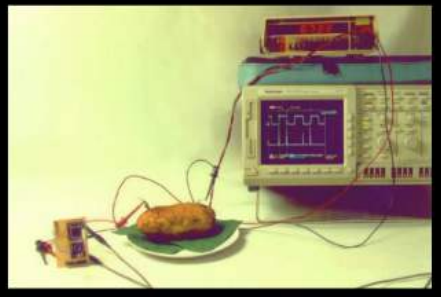
(c) Silirium.ru





Asynchronous Microprocessors, East and West



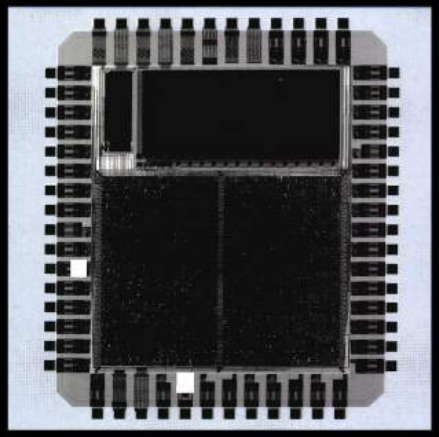


Soviet bit-slice microprocessor modules (late 1970s) produced as

- K587 (Elektronika NC microarchitecture core, with PDP-11 compatible instruction set)
- K588 (PDP-11 instruction set)
- K1883 (controller, also known as U83x in East Germany)

The Caltech Asynchronous Microprocessor (CAM), the world-first asynchronous microprocessor (1988).

The Philips Semiconductors asynchronous 80C51 microcontroller.



Initially aimed for use in pager chipsets, and the motivation was to lower electromagnetic interference (EMI) noise emissions so that the microcontroller could operate harmoniously with the radio-frequency (RF) data link, without the use of shielding.

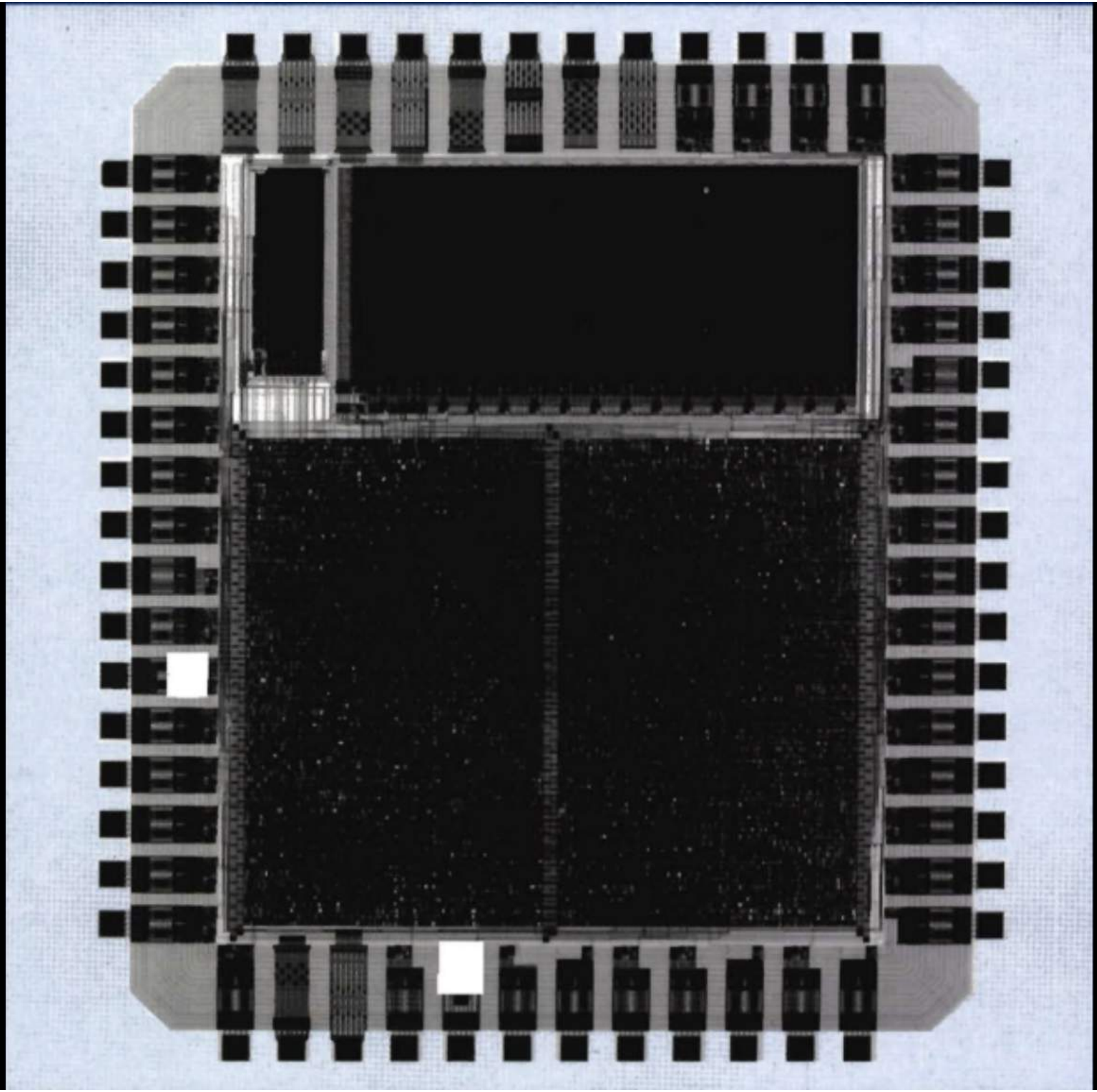
Contains 128 bytes of RAM , 32 I/O lines, three 16-bit counter/timers, a six-source, and a four-priority level nested interrupt structure.

Demonstrated a 4x power reduction over comparable synchronous microcontrollers.

later used in public transport smart cards, because it could be powered by only the brief burst of energy induced by a wave through the card reader.

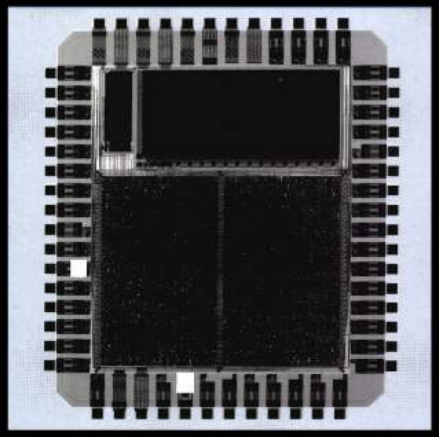
the enhanced SmartMX microcontroller is now used in more than 75 countries, including the EU and the US, for biometric passports, IDs, electronic toll collection, car parking, and loyalty programs.

By 2013, the number of copies of the SmartMX (and variants) sold has exceeded 2 billion chips.




T=CL card 04
7417C9922680
MIFARE
NXP

The Philips Semiconductors asynchronous 80C51 microcontroller.



Initially aimed for use in pager chipsets, and the motivation was to lower electromagnetic interference (EMI) noise emissions so that the microcontroller could operate harmoniously with the radio-frequency (RF) data link, without the use of shielding.

Contains 128 bytes of RAM , 32 I/O lines, three 16-bit counter/timers, a six-source, and a four-priority level nested interrupt structure.

Demonstrated a 4x power reduction over comparable synchronous microcontrollers.

later used in public transport smart cards, because it could be powered by only the brief burst of energy induced by a wave through the card reader.

the enhanced SmartMX microcontroller is now used in more than 75 countries, including the EU and the US, for biometric passports, IDs, electronic toll collection, car parking, and loyalty programs.

By 2013, the number of copies of the SmartMX (and variants) sold has exceeded 2 billion chips.

Intel/Fulcrum Microsystems: Ethernet Switch Chips

In 2011, Intel acquired Fulcrum Microsystems in a move regarded as a bid to compete with Cisco Systems.

Intel's current FM5000/FM6000 family of switch chips supports 40 gigabit Ethernet.

Includes a fully-asynchronous high-speed crossbar switch that provides high bandwidth, low latency, support for flexible link topologies, and high energy efficiency.

The crossbar bandwidth of over 1 terabit per second (in a 130 nm process) is achieved through fine-grain asynchronous pipelining, at the granularity of individual gates, unencumbered by a rigid clock period.

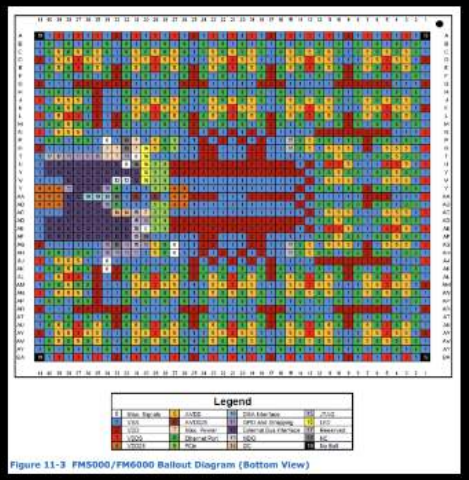
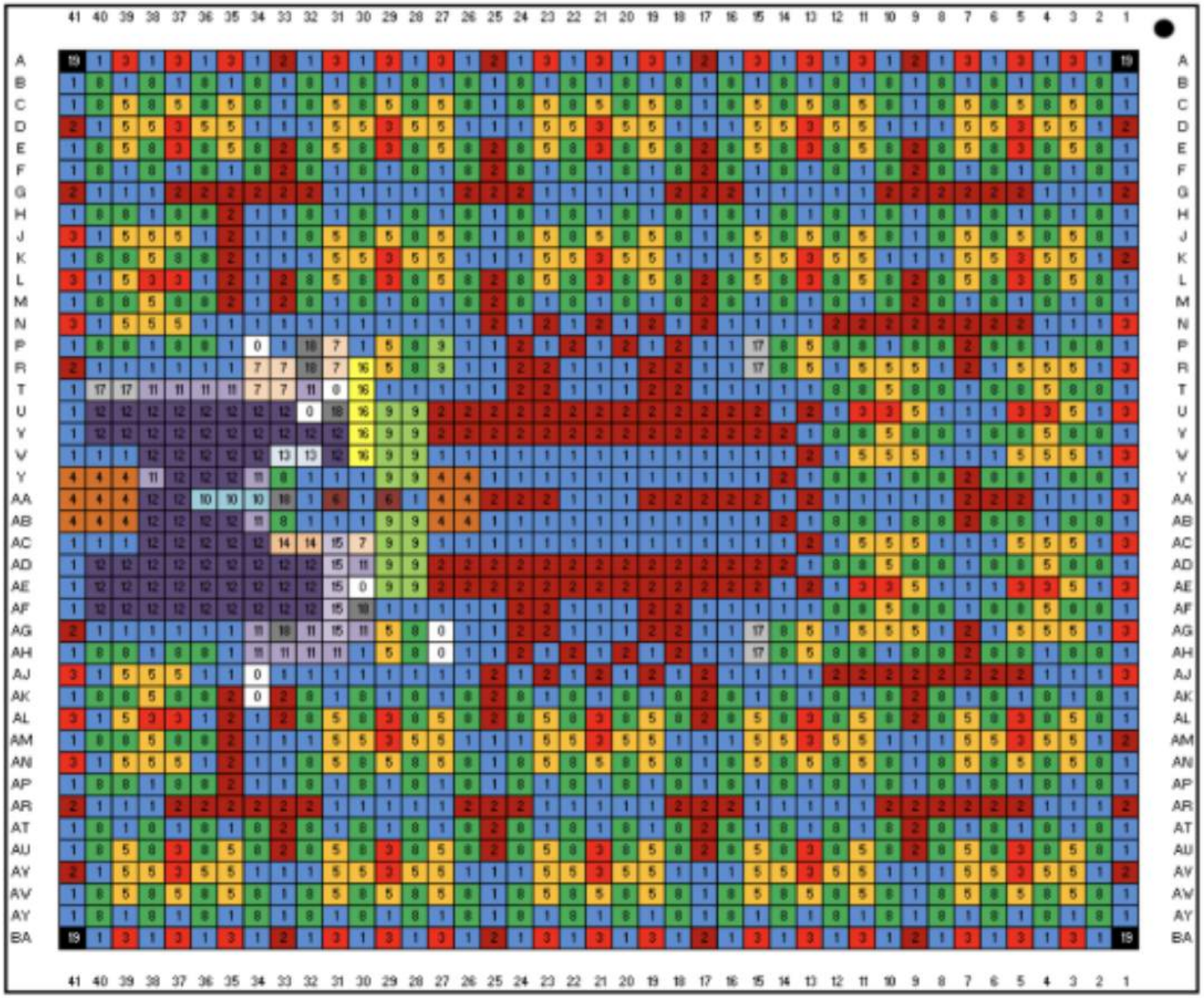


Figure 11-3 FM5000/FM6000 Ballout Diagram (Bottom View)


12
W340FT03
EZFM5224A



Legend							
0	Misc. Signals	5	AVDD	10	DMA Interface	15	JTAG
1	VSS	6	AVDD25	11	GPIO and Strapping	16	LED
2	VDD	7	Misc. Power	12	External Bus Interface	17	Reserved
3	VDDS	8	Ethernet Port	13	MDIO	18	NC
4	VDD25	9	PCIe	14	I2C	19	No Ball

Figure 11-3 FM5000/FM6000 Ballout Diagram (Bottom View)

Intel/Fulcrum Microsystems: Ethernet Switch Chips

In 2011, Intel acquired Fulcrum Microsystems in a move regarded as a bid to compete with Cisco Systems.

Intel's current FM5000/FM6000 family of switch chips supports 40 gigabit Ethernet.

Includes a fully-asynchronous high-speed crossbar switch that provides high bandwidth, low latency, support for flexible link topologies, and high energy efficiency.

The crossbar bandwidth of over 1 terabit per second (in a 130 nm process) is achieved through fine-grain asynchronous pipelining, at the granularity of individual gates, unencumbered by a rigid clock period.

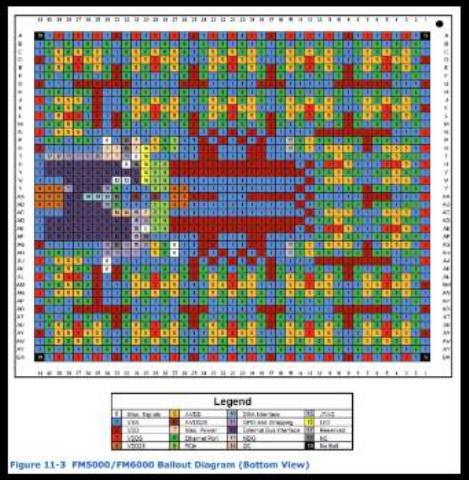


Figure 11-3 FM5000/FM6000 Ballout Diagram (Bottom View)

Achronix: high-performance FPGAs

A FPGA (Field Programmable Date Array) is an integrated circuit designed to be configured after manufacture.

The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC).

The Speedster 22i family of FPGAs are manufactured in 22 nm.

They operate at 1.5 GHz and are currently claimed as the world's fastest FPGAs.


achronix
SPEEDSTER®
22i
AC22IHD1000-F53C1
FPO1234567
USA
VCC 5V
3P3V
GND
GND

Achronix: high-performance FPGAs

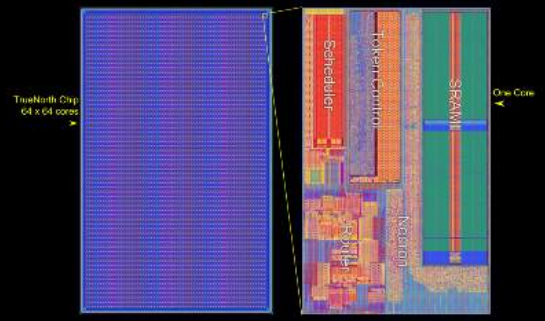
A FPGA (Field Programmable Date Array) is an integrated circuit designed to be configured after manufacture.

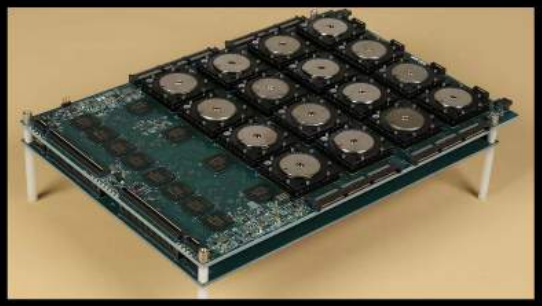
The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC).

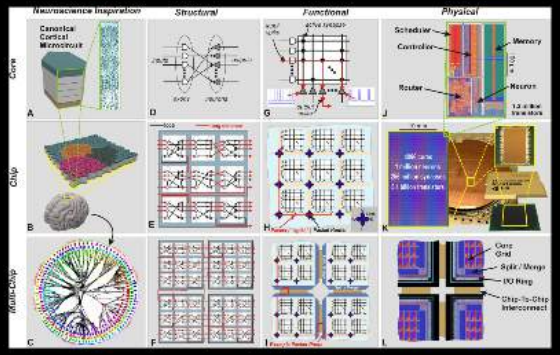
The Speedster 22i family of FPGAs are manufactured in 22 nm.

They operate at 1.5 GHz and are currently claimed as the world's fastest FPGAs.

The IBM TrueNorth Neuromorphic Computer







There has been much excitement recently about neuromorphic computing, which seeks to mimic the functioning of the human brain by using massively- parallel computer systems.

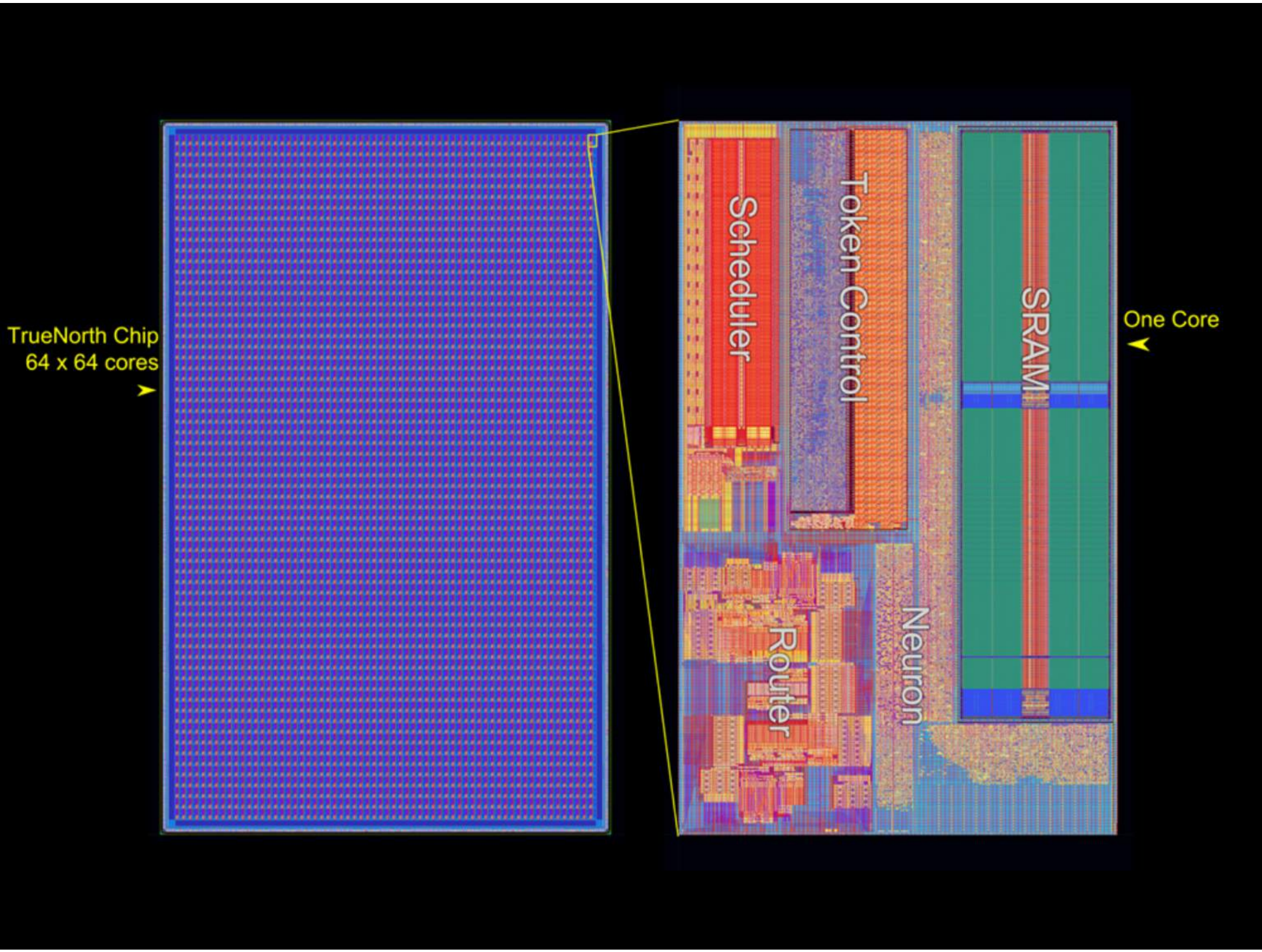
A departure from the traditional von Neumann architecture, these systems employ a highly-distributed memory that is tightly integrated with a large number of parallel computational elements that model neurons.

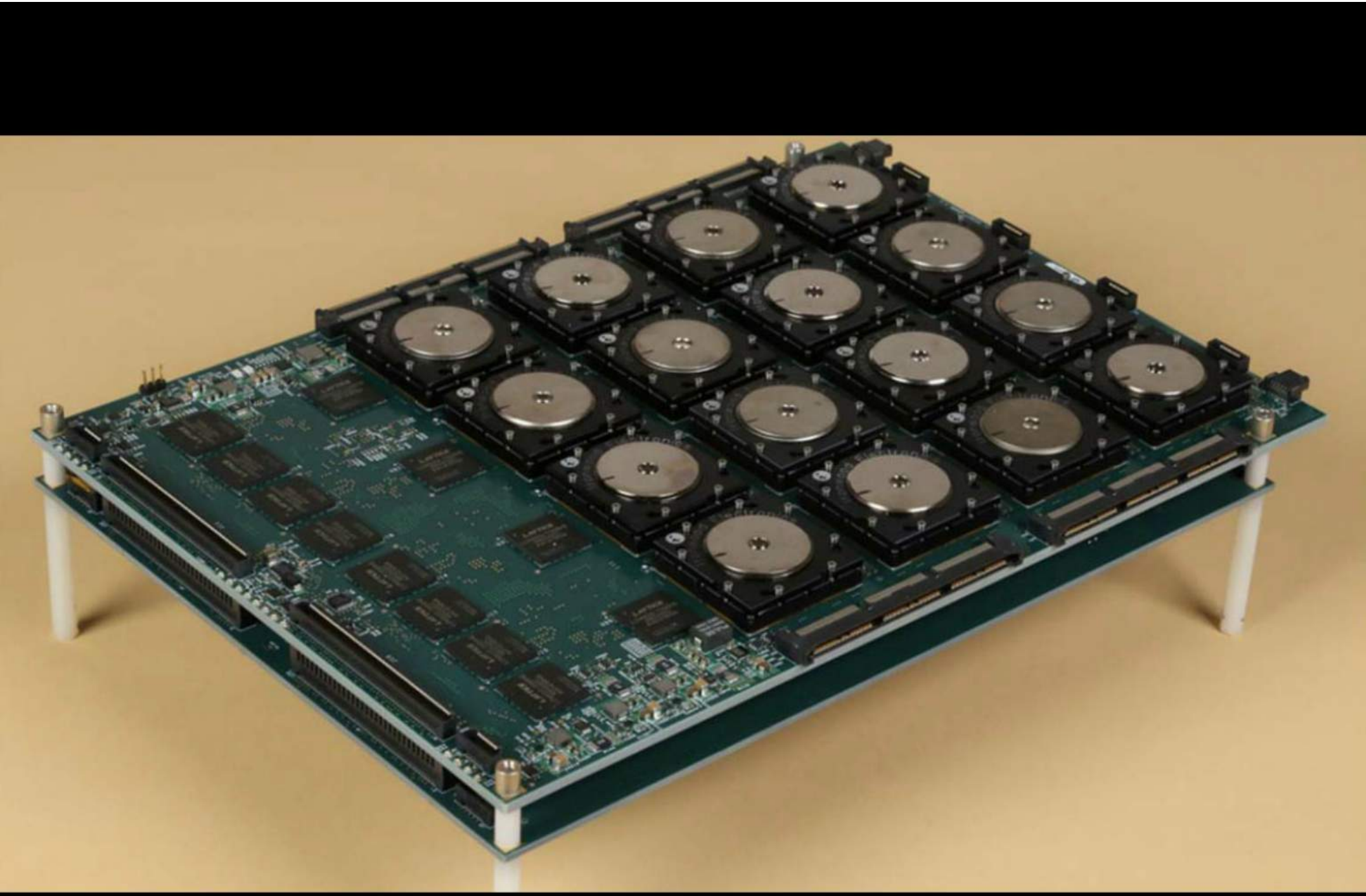
Due to the spatially-distributed nature of computation and communication, along with wide timing unpredictability of data events, neuromorphic computing fits well with the asynchronous paradigm.

Released in August 2014, which is the largest chip ever developed at IBM, with 5.4 billion transistors.

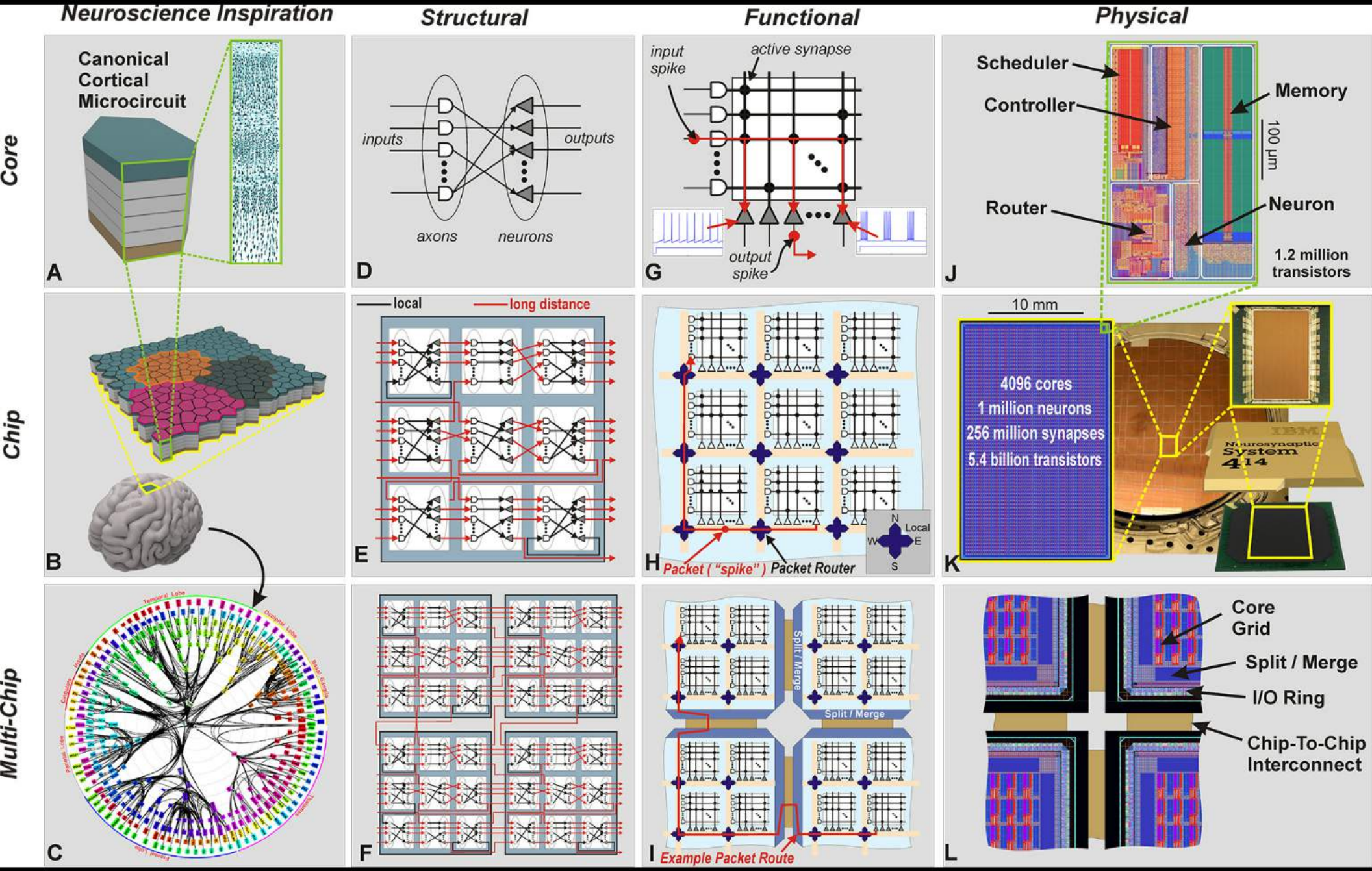
It integrates 4096 neurosynaptic cores on a single chip, modelling 1 million neurons and 256 million synapses.

The event-driven asynchronous operation facilitates extremely low power - 70 mW for real-time operation over the entire chip - which would be extremely difficult to obtain with current synchronous techniques.

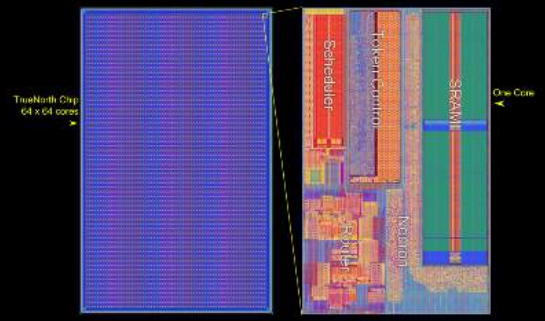

TrueNorth Chip
64 x 64 cores
Scheduler
Token Control
SRAM
Router
Neuron
One Core

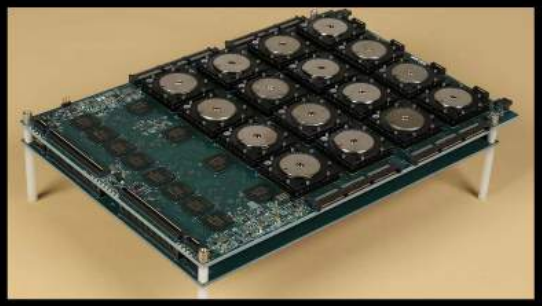


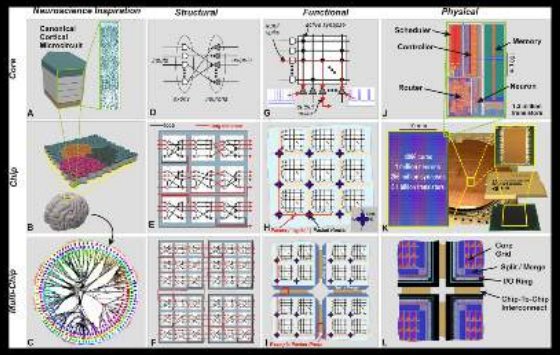
Neuroscience Inspiration	Structural	Functional	Physical
		input active synapse	


Neuroscience Inspiration
Structural
Functional
Physical
Core
Chip
Multi-Chip
Canonical Cortical Microcircuit
A
inputs
outputs
axons
neurons
D
input spike
active synapse
output spike
G
Scheduler
Controller
Memory
100 um
Router
Neuron
1.2 million transistors
J
B
local
long distance
E
10 mm
4096 cores
1 million neurons
256 million synapses
5.4 billion transistors
IBM
Neurosynaptic System
4x4
K
N
Local
W
E
S
H
Packet ("spike")
Packet Router
C
F
Split / Merge
Split / Merge
I
Example Packet Route
Core Grid
Split / Merge
I/O Ring
Chip-To-Chip Interconnect
L

The IBM TrueNorth Neuromorphic Computer







There has been much excitement recently about neuromorphic computing, which seeks to mimic the functioning of the human brain by using massively- parallel computer systems.

A departure from the traditional von Neumann architecture, these systems employ a highly-distributed memory that is tightly integrated with a large number of parallel computational elements that model neurons.

Due to the spatially-distributed nature of computation and communication, along with wide timing unpredictability of data events, neuromorphic computing fits well with the asynchronous paradigm.

Released in August 2014, which is the largest chip ever developed at IBM, with 5.4 billion transistors.

It integrates 4096 neurosynaptic cores on a single chip, modelling 1 million neurons and 256 million synapses.

The event-driven asynchronous operation facilitates extremely low power - 70 mW for real-time operation over the entire chip - which would be extremely difficult to obtain with current synchronous techniques.

Oh, BTW, Computers and Brains Aren't Too Much Alike

Brains are analogue, while computers are digital.

Brains are massively parallel, while computers have a very long way to go in that regard.

Brains use content-addressable memory, while very few computers have a good handle on that.

Short-term memory in brains seems to form "pointers" to long-term memory, while computer RAM is much more like a byte for byte copy of persistent memory on disks and the like.

There is no distinction between hardware and software in the brain: changes in the mind change the brain, but in computers, we abstract away the hardware from the software.

Neurons and synapses are much more complex than electrical logic gates in computers - signals on axons are actually electrochemical in nature.

Brains do not operate like Von Neumann computers: processing and memory are the done by the same components in the brain (there is no CPU).

Brains have natural neural plasticity built in, whereas most computers have little to offer in the form of self-healing behavior.

Brains have bodies, which can offer unique advantages. For example, despite your intuitive feeling that you could close your eyes and know the locations of objects around you, a series of experiments in the field of change blindness has shown that our visual memories are actually quite sparse.

What's happening is that the brain is "offloading" its memory requirements to the environment in which it exists: why bother remembering the location of objects when a quick glance will suffice?

Brains are much bigger than even the largest computers to date.

Accurate biological models of the brain would have to include some 225,000,000,000,000,000 (225 million billion) interactions between cell types, neurotransmitters, neuromodulators, axonal branches and dendritic spines.

Brains are analogue, while computers are digital.

Brains are massively parallel, while computers have a very long way to go in that regard.

Brains use content-addressable memory, while very few computers have a good handle on that.

Short-term memory in brains seems to form "pointers" to long-term memory, while computer RAM is much more like a byte for byte copy of persistent memory on disks and the like.

There is no distinction between hardware and software in the brain: changes in the mind change the brain, but in computers, we abstract away the hardware from the software.

Neurons and synapses are much more complex than

Hardware from the software.

Neurons and synapses are much more complex than electrical logic gates in computers - signals on axons are actually electrochemical in nature.

Brains do not operate like Von Neumann

Brains do not operate like Von Neumann computers: processing and memory are the done by the same components in the brain (there is no CPU).

CPU).

Brains have natural neural plasticity built in, whereas most computers have little to offer in the form of self-healing behavior.

Brains have bodies, which can offer unique advantages. For example, despite your intuitive feeling that you could close your eyes and know the locations of objects around you, a series of experiments in the field of change blindness has shown that our visual memories are actually quite sparse.

What's happening is that the brain is "offloading" its memory requirements to the environment in which it exists: why bother remembering the location of objects when a quick glance will suffice?

Brains are much bigger than even the largest computers to date.

Accurate biological models of the brain would have to include some 225,000,000,000,000,000 (225 million billion) interactions between cell types, neurotransmitters, neuromodulators, axonal branches and dendritic spines.

Oh, BTW, Computers and Brains Aren't Too Much Alike

Brains are analogue, while computers are digital.

Brains are massively parallel, while computers have a very long way to go in that regard.

Brains use content-addressable memory, while very few computers have a good handle on that.

Short-term memory in brains seems to form "pointers" to long-term memory, while computer RAM is much more like a byte for byte copy of persistent memory on disks and the like.

There is no distinction between hardware and software in the brain: changes in the mind change the brain, but in computers, we abstract away the hardware from the software.

Neurons and synapses are much more complex than electrical logic gates in computers - signals on axons are actually electrochemical in nature.

Brains do not operate like Von Neumann computers: processing and memory are the done by the same components in the brain (there is no CPU).

Brains have natural neural plasticity built in, whereas most computers have little to offer in the form of self-healing behavior.

Brains have bodies, which can offer unique advantages. For example, despite your intuitive feeling that you could close your eyes and know the locations of objects around you, a series of experiments in the field of change blindness has shown that our visual memories are actually quite sparse.

What's happening is that the brain is "offloading" its memory requirements to the environment in which it exists: why bother remembering the location of objects when a quick glance will suffice?

Brains are much bigger than even the largest computers to date.

Accurate biological models of the brain would have to include some 225,000,000,000,000,000 (225 million billion) interactions between cell types, neurotransmitters, neuromodulators, axonal branches and dendritic spines.

Famous Asynchronous Digital Logic Devices That You May Be Familiar With

The ILLIAC I - 1952

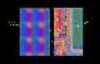
Asynchronous Microprocessors, East and West

The Philips Semiconductors asynchronous 80C51 microcontroller

Intel/Fulcrum Microsystems: Ethernet Switch Chips

Achronix: high-performance FPGAs

The IBM TrueNorth Neuromorphic Computer



Oh, BTW, Computers and Brains Aren't Too Much Alike

WTF is this talk about?

A Tiny Bit of Theory About Digital Electronics - Synchronous vs Asynchronous

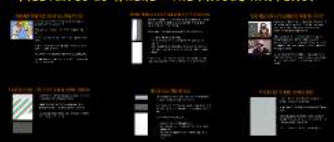
Famous Asynchronous Digital Logic Devices That You May Be Familiar With

Why Asynchronous is a Hot Topic in Digital Logic Design Today

How and Why are Organizations Becoming More Like Synchronous Digital Circuits?

Are You Saying That All the SAFe "Flow Model" Stuff is All Wrong?

Where did organizations get the idea that meetings is where "the magic happens?"



OK. Now, as an Organization, Let's Improve and Reduce Meetings. What Would That Look Like?

On Virtualization and Containerization and Why They Can Help

Closing Thoughts and Takeaways

Why Asynchronous is a Hot Topic in Digital Logic Design Today

Moore's Law Has Stalled, But Asynchronous Can Save Us

simple Von Neumann computers winning against ever faster can't happen anymore because of the effect of physics on element size and faster clock speeds.

But with asynchronous circuits, we can continue into the future:

- No worries about clock skew (the need to send a single clock to all of the hardware's circuits at once - as clock speed gets faster, but the speed of light stays the same. But with asynchronous circuits we don't have to worry about a clock that needs to be globally distributed and we don't have to slow down a circuit to help accommodate the skew).
- Lower power requirements (we only have to worry about those portions of the circuit that are in use for the current computation, and the unused portions of the circuit don't get pointed unnecessarily).
- Average-case instead of worst-case performance (Synchronous circuits must wait until all possible computations have completed before latching the results, yielding worst-case performance. Many asynchronous systems sense when a computation has completed, allowing them to exhibit average-case performance.)
- Easing of global timing issues (In systems such as a synchronous microprocessor, the system clock, and thus system performance, is dictated by the slowest (critical) path. Thus, most portions of a circuit must be carefully optimized to achieve the highest clock rate, including rarely used portions of the system. Since many asynchronous systems operate at the speed of the circuit path currently in operation, rarely used portions of the circuits can be left un-optimized without adversely affecting system performance.)
- Automatic adaptation to physical properties (The delay through a circuit can change with variations in fabrication, temperature, and power supply voltage. Synchronous circuits must assume that the worst possible combination of factors is present and clock the system accordingly. Many asynchronous circuits sense computation completion, and will run as quickly as the current physical properties allow.)

So, Why Are So Many New Chips Synchronous?

The reasons I think that Asynchronous isn't commonplace today:

- Design tools.
- Most CPU design tools assume a clocked CPU.
- Most tools enforce synchronous design practices.
- Making a clockless CPU involves modifying the design tools to handle clockless logic and doing extra testing to ensure the design avoids metastable problems.
- And most CPU designers have been trained to design synchronous circuits these days!

It's just easier to design synchronous circuits given where we are. So that's what we mostly do.

OK. Enough about electronics. Back to software development.

Moore's Law Has Stalled, But Asynchronous Can Save Us

simple Von Neumann computers running against ever faster can't happen anymore because of the effect of physics on element size and faster clock speeds..

But with asynchronous circuits, we can continue into the future:

- No worries about clock skew (the need to send a single clock to all of the machine's circuits at once - as clock speed gets faster but the speed of light stays the same. But with asynchronous circuits we don't have to worry about a clock that needs to be globally distributed and we don't have to slow down a circuit to help accommodate the skew).
- Lower power requirements (we only have to worry about those potions of the circuit that are in use for the current computation, and the unused portions of the circuit don't get powered unnecessarily).
- Average-case instead of worst-case performance (Synchronous circuits must wait until all possible computations have completed before latching the results, yielding worst-case performance. Many asynchronous systems sense when a computation has completed, allowing them to exhibit average-case performance.)
- Easing of global timing issues (In systems such as a synchronous microprocessor, the system clock, and thus system performance, is dictated by the slowest (critical) path. Thus, most portions of a circuit must be carefully optimized to achieve the highest clock rate, including rarely used portions of the system. Since many asynchronous systems operate at the speed of the circuit path currently in operation, rarely used portions of the circuit can be left un-optimized without adversely affecting system performance.)
- Automatic adaptation to physical properties (The delay through a circuit can change with variations in fabrication, temperature, and power-supply voltage. Synchronous circuits must assume that the worst possible combination of factors is present and clock the system accordingly. Many asynchronous circuits sense computation completion, and will run as quickly as the current physical properties allow.)

So, Why Are So Many New Chips Synchronous?

The reasons I think that Asynchronous isn't commonplace today:

- Design tools.
- most CPU design tools assume a clocked CPU.
- most tools enforce synchronous design practices.
- making a clockless CPU involves modifying the design tools to handle clockless logic and doing extra testing to ensure the design avoids metastable problems.
- And most CPU designers have been trained to design synchronous circuits these days!

It's just easier to design synchronous circuits given where we are. So that's what we mostly do.

OK. Enough about electronics. Back to software development.

Why Asynchronous is a Hot Topic in Digital Logic Design Today

Moore's Law Has Stalled, But Asynchronous Can Save Us

simple Von Neumann computers winning against ever faster can't happen anymore because of the effect of physics on element size and faster clock speeds.

But with asynchronous circuits, we can continue into the future:

- No worries about clock skew (the need to send a single clock to all of the hardware's circuits at once - as clock speed gets faster, but the speed of light stays the same. But with asynchronous circuits we don't have to worry about a clock that needs to be globally distributed and we don't have to slow down a circuit to help accommodate the skew).
- Lower power requirements (we only have to worry about those portions of the circuit that are in use for the current computation, and the unused portions of the circuit don't get pointed unnecessarily).
- Average-case instead of worst-case performance (Synchronous circuits must wait until all possible computations have completed before latching the results, yielding worst-case performance. Many asynchronous systems sense when a computation has completed, allowing them to exhibit average-case performance.)
- Easing of global timing issues (In systems such as a synchronous microprocessor, the system clock, and thus system performance, is dictated by the slowest (critical) path. Thus, most portions of a circuit must be carefully optimized to achieve the highest clock rate, including rarely used portions of the system. Since many asynchronous systems operate at the speed of the circuit path currently in operation, rarely used portions of the circuits can be left un-optimized without adversely affecting system performance.)
- Automatic adaptation to physical properties (The delay through a circuit can change with variations in fabrication, temperature, and power supply voltage. Synchronous circuits must assume that the worst possible combination of factors is present and clock the system accordingly. Many asynchronous circuits sense computation completion, and will run as quickly as the current physical properties allow.)

So, Why Are So Many New Chips Synchronous?

The reasons I think that Asynchronous isn't commonplace today:

- Design tools.
- Most CPU design tools assume a clocked CPU.
- Most tools enforce synchronous design practices.
- Making a clockless CPU involves modifying the design tools to handle clockless logic and doing extra testing to ensure the design avoids metastable problems.
- And most CPU designers have been trained to design synchronous circuits these days!

It's just easier to design synchronous circuits given where we are. So that's what we mostly do.

OK. Enough about electronics. Back to software development.

WTF is this talk about?

A Tiny Bit of Theory About Digital Electronics - Synchronous vs Asynchronous

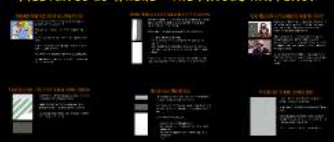
Famous Asynchronous Digital Logic Devices That You May Be Familiar With

Why Asynchronous is a Hot Topic in Digital Logic Design Today

How and Why are Organizations Becoming More Like Synchronous Digital Circuits?

Are You Saying That All the SAFe "Flow Model" Stuff is All Wrong?

Where did organizations get the idea that meetings is where "the magic happens?"



OK. Now, as an Organization, Let's Improve and Reduce Meetings. What Would That Look Like?

On Virtualization and Containerization and Why They Can Help

Closing Thoughts and Takeaways

How and Why are Organizations Becoming More Like Synchronous Digital Circuits?

Because It's a Reaction to Complicated Things

Organizations have taken the same play that Synchronous Circuit Designers took when things got complicated.

Fear of uncertainty, especially in risk adverse organizations that were afraid that they couldn't do "bigger things" made them decide that they needed to "Scale the Organization" or "Scale Agile" to accomplish these things.

Because the Concept is Simple

And just like with Synchronous Circuit Designers, the use of clocks is a fairly easy to understand concept.

It works in exactly the same way: when the clock signal happens, we use whatever information is available on our inputs and get to the next state.

We just call it different things in a corporate setting.

Like PI Planning, meetings, Agile Process tools, and Agile Release Trains.

Because It's a Reaction to Complicated Things

Organizations have taken the same play that Synchronous Circuit Designers took when things got complicated.

Fear of uncertainty, especially in risk adverse organizations that were afraid that they couldn't do "bigger things" made them decide that they needed to "Scale the Organization" or "Scale Agile" to accomplish these things.

Because the Concept is Simple

And just like with Synchronous Circuit Designers, the use of clocks is a fairly easy to understand concept.

It works in exactly the same way: when the clock signal happens, we use whatever information is available on our inputs and get to the next state.

We just call it different things in a corporate setting.

Like PI Planning, meetings, Agile Process tools, and Agile Release Trains.

How and Why are Organizations Becoming More Like Synchronous Digital Circuits?

Because It's a Reaction to Complicated Things

Organizations have taken the same play that Synchronous Circuit Designers took when things got complicated.

Fear of uncertainty, especially in risk adverse organizations that were afraid that they couldn't do "bigger things" made them decide that they needed to "Scale the Organization" or "Scale Agile" to accomplish these things.

Because the Concept is Simple

And just like with Synchronous Circuit Designers, the use of clocks is a fairly easy to understand concept.

It works in exactly the same way: when the clock signal happens, we use whatever information is available on our inputs and get to the next state.

We just call it different things in a corporate setting.

Like PI Planning, meetings, Agile Process tools, and Agile Release Trains.

WTF is this talk about?

A Tiny Bit of Theory About Digital Electronics - Synchronous vs Asynchronous

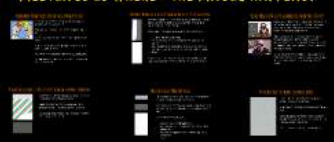
Famous Asynchronous Digital Logic Devices That You May Be Familiar With

Why Asynchronous is a Hot Topic in Digital Logic Design Today

How and Why are Organizations Becoming More Like Synchronous Digital Circuits?

Are You Saying That All the SAFe "Flow Model" Stuff is All Wrong?

Where did organizations get the idea that meetings is where "the magic happens?"



OK. Now, as an Organization, Let's Improve and Reduce Meetings. What Would That Look Like?

On Virtualization and Containerization and Why They Can Help

Closing Thoughts and Takeaways

Are You Saying That All the SAFe "Flow Model" Stuff is All Wrong?

Yeah. Much of it is wrong. But it's a good business!

I had to go to the doctor recently. Actually my gastroenterologist told me that he was out of ideas, and referred me to the "Mecca" for my condition. Mount Sinai Hospital in Manhattan.

I met a doctor there that was, bar none, the best doctor I've ever worked with. A rare combination of a very smart person who also has great bedside manner. He's actually a professor at the School of Medicine who also sees some patients.

And then I realized something about doctors. There are three types:

- Those who research, write papers, and present at conferences (like this new doctor).
- Those who read the occasional paper on topics they deal with (like my current doctor)
- Those who are licensed, but only learn of new things through pharmaceutical reps and product brochures.

And that's when I realized that it occurs in software development as well.

Let's Look for Clocks in This Popular "Scaled Agile Framework"

[illegible]

[illegible]

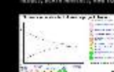
[illegible]

[illegible]

And Here's the two extremely damming aspects to all of this

While SAFe is sold as "Scaling Agile to the Enterprise", the reality is that once we reach Dunbar's Number (around 150 people), you can no longer have effective PI planning meetings. There's just too many people to interact with effectively. This is a direct analogy to clock fanout problems on synchronous circuits where things like clock skew must be taken into account, and the root cause of why it's really tough to scale clocked CPUs past certain clock speeds.

Since SAFe requires commitment between development to do all that planning with, and the errors in estimating software development time are way too variable to give good guidance in planning, the work committed to never matches the reality of the development process, and making the dates still requires heroics, death marches, and compromising on quality.



Here's the Real Problem to Solve

I want organizations to stop trying to scale up team Agile practices to work on the projects that are "Enterprise Big" and instead to work on scaling down those big monster monolithic systems into smaller loosely coupled systems of highly cohesive parts, and be smart about how the parts get wired together.

Ok. Technically, one method of accomplishing this is to use microservices that work over versioned APIs that are tolerant of version mismatches, can automatically fall back to previous API versions, and anneal themselves by real time negotiation of versioning.

Yes. That will take software craftsmanship and major architectural refactoring for the majority of the monolithic systems that I encounter. (It's my plight that I'm never involved with the organizations that have these sorts of architectures in place already. I guess they have no need for me, since they are already doing the right things. :-))

And What About ARTs, AKA CI Pipelines

This is nothing more than traditional DTAP (Development, Test, Accept, Production) thinking, with decision making, hand wringing, and wait states for approvals to occur.

Most of the Agile Release Train Wrecks occur when one DTAP pipeline does not pump sufficiently pure code for the integration pipelines to consume. The train's engines sputter and die horrible deaths instead of getting to where they needed to be.

You should always strive to develop full stack on trunk and deploy upon positive automated test results.

Releases should not be viewed as something abnormal and risky. Releases should happen all the time, because that's why we develop code in the first place.

Yeah. Much of it is wrong. But it's a good business!

I had to go to the doctor recently. Actually my gastroenterologist told me that he was out of ideas, and referred me to the "Mecca" for my condition. Mount Sinai Hospital in Manhattan.

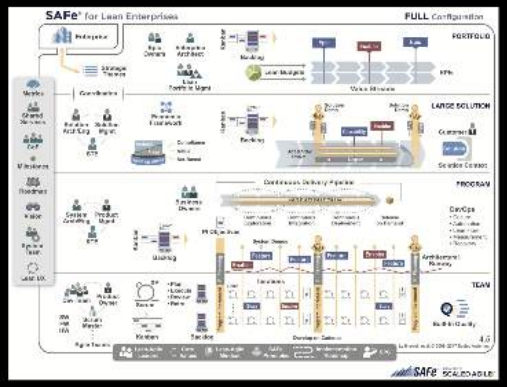
I met a doctor there that was, bar none, the best doctor I've ever worked with. A rare combination of a very smart person who also has great bedside manner. He's actually a professor at the School of Medicine who also sees some patients.

And then I realized something about doctors. There are three types:

- Those who research, write papers, and present at conferences (like this new doctor).
- Those who read the occasional paper on topics they deal with (like my current doctor)
- Those who are licensed, but only learn of new things through pharmaceutical reps and product brochures.

And that's when I realized that it occurs in software development as well.

Let's Look for Clocks in This Popular "Scaled Agile Framework"



Do you see the clocks? They're hiding behind "Agile/Lean/Scrum" sounding words. But they're there! Let's take a look.

Within teams, Scrum is supposed to be a savior. The traditional Scrum ceremonies have daily stand-ups, bi-weekly sprint planning, bi-weekly reviews/demos, bi-weekly retrospectives, bi-weekly grooming, etc. Teams then work in synchronized sprints, so they can have synchronized system demos.

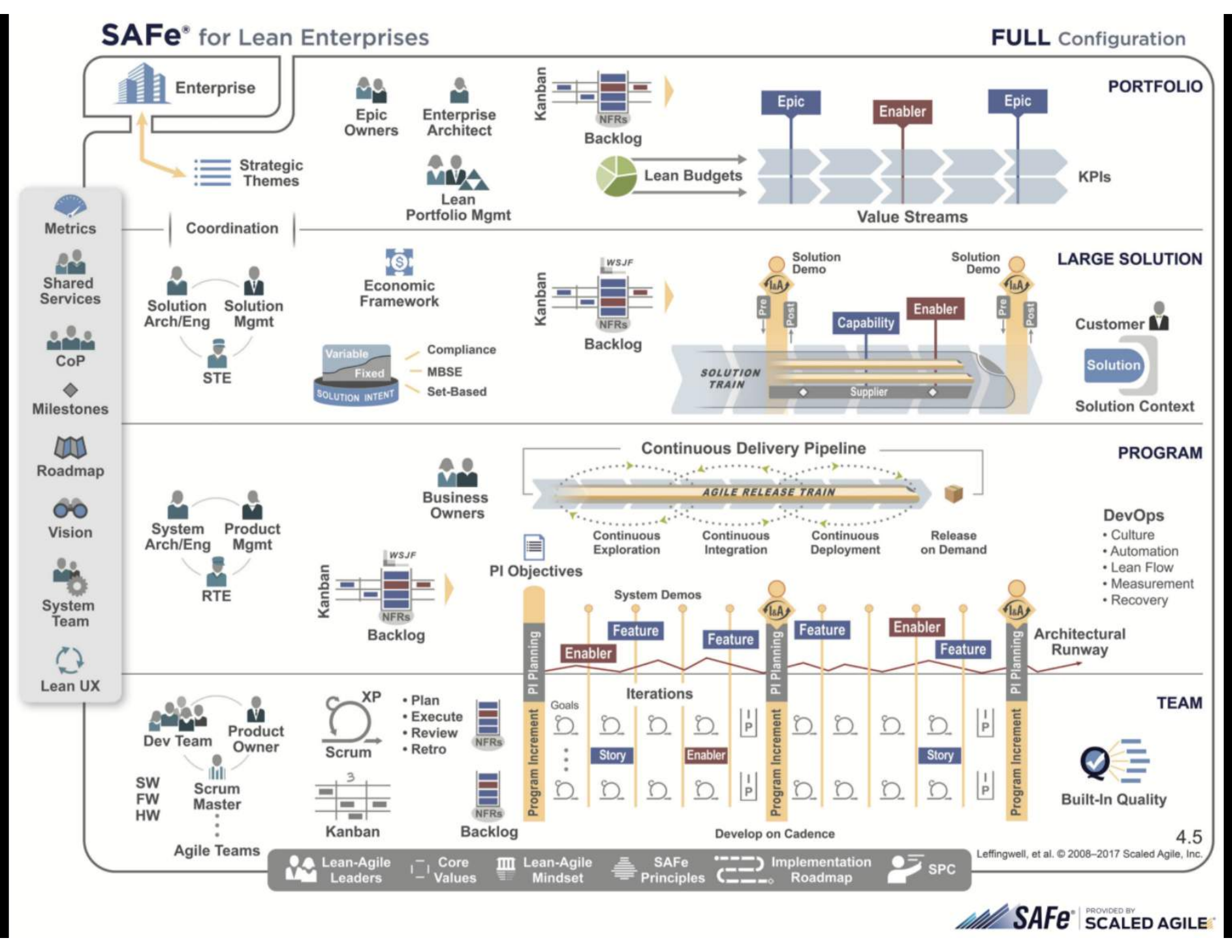
PI planning happens perhaps every quarter. These can be huge events, typically taking 2 days with all synchronized teams attending. [The PI events amaze me the most, as that's where we figure out the dependencies between various sections of our codebase and plan the best order. I say, "don't waste time try to plan your dependencies. Instead, spend the time to break the dependencies with good architecture and code!]

Then, we have the "Agile Release Train" concept. Which is supposed to happen every day or week, but typically happens right before the quarterly releases, so the "hardening sprints" can occur. [The whole concept of Release Trains is that if you miss one, another one is coming soon, so don't fret. But in reality for organizations that are still stuck in large monolithic architectures, we have heroes and death marches happening to assemble the trains, which usually don't end in released code, but train wrecks!]

And less obvious in this diagram, because it's shrouded in the non-commitment driven development word called "Kanban", we see that management has budget cycles, coordination on solution engineering, and KPI measurements (Key Performance Indicators, which are time based and usually tied to some clock that's running)

SAFe® for Lean Enterprises

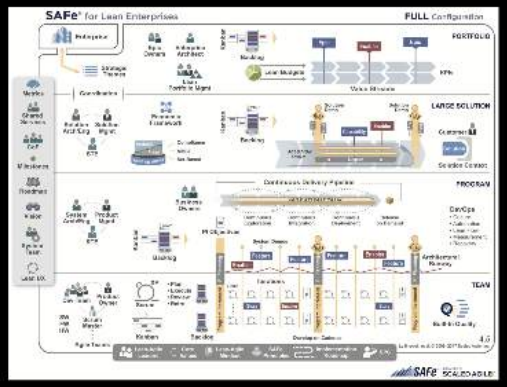
FULL Configuration



Leffingwell, et al. © 2008–2017 Scaled Agile, Inc.

SAFe® PROVIDED BY SCALED AGILE

Let's Look for Clocks in This Popular "Scaled Agile Framework"



Do you see the clocks? They're hiding behind "Agile/Lean/Scrum" sounding words. But they're there! Let's take a look.

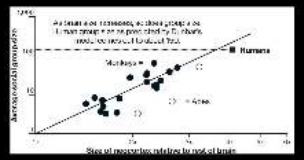
Within teams, Scrum is supposed to be a savior. The traditional Scrum ceremonies have daily stand-ups, bi-weekly sprint planning, bi-weekly reviews/demos, bi-weekly retrospectives, bi-weekly grooming, etc. Teams then work in synchronized sprints, so they can have synchronized system demos.

PI planning happens perhaps every quarter. These can be huge events, typically taking 2 days with all synchronized teams attending. [The PI events amaze me the most, as that's where we figure out the dependencies between various sections of our codebase and plan the best order. I say, "don't waste time try to plan your dependencies. Instead, spend the time to break the dependencies with good architecture and code!]

Then, we have the "Agile Release Train" concept. Which is supposed to happen every day or week, but typically happens right before the quarterly releases, so the "hardening sprints" can occur. [The whole concept of Release Trains is that if you miss one, another one is coming soon, so don't fret. But in reality for organizations that are still stuck in large monolithic architectures, we have heroes and death marches happening to assemble the trains, which usually don't end in released code, but train wrecks!]

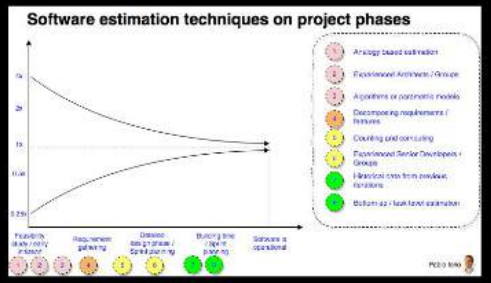
And less obvious in this diagram, because it's shrouded in the non-commitment driven development word called "Kanban", we see that management has budget cycles, coordination on solution engineering, and KPI measurements (Key Performance Indicators, which are time based and usually tied to some clock that's running)

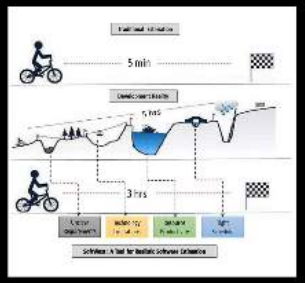
And Here's the two extremely damming aspects to all of this

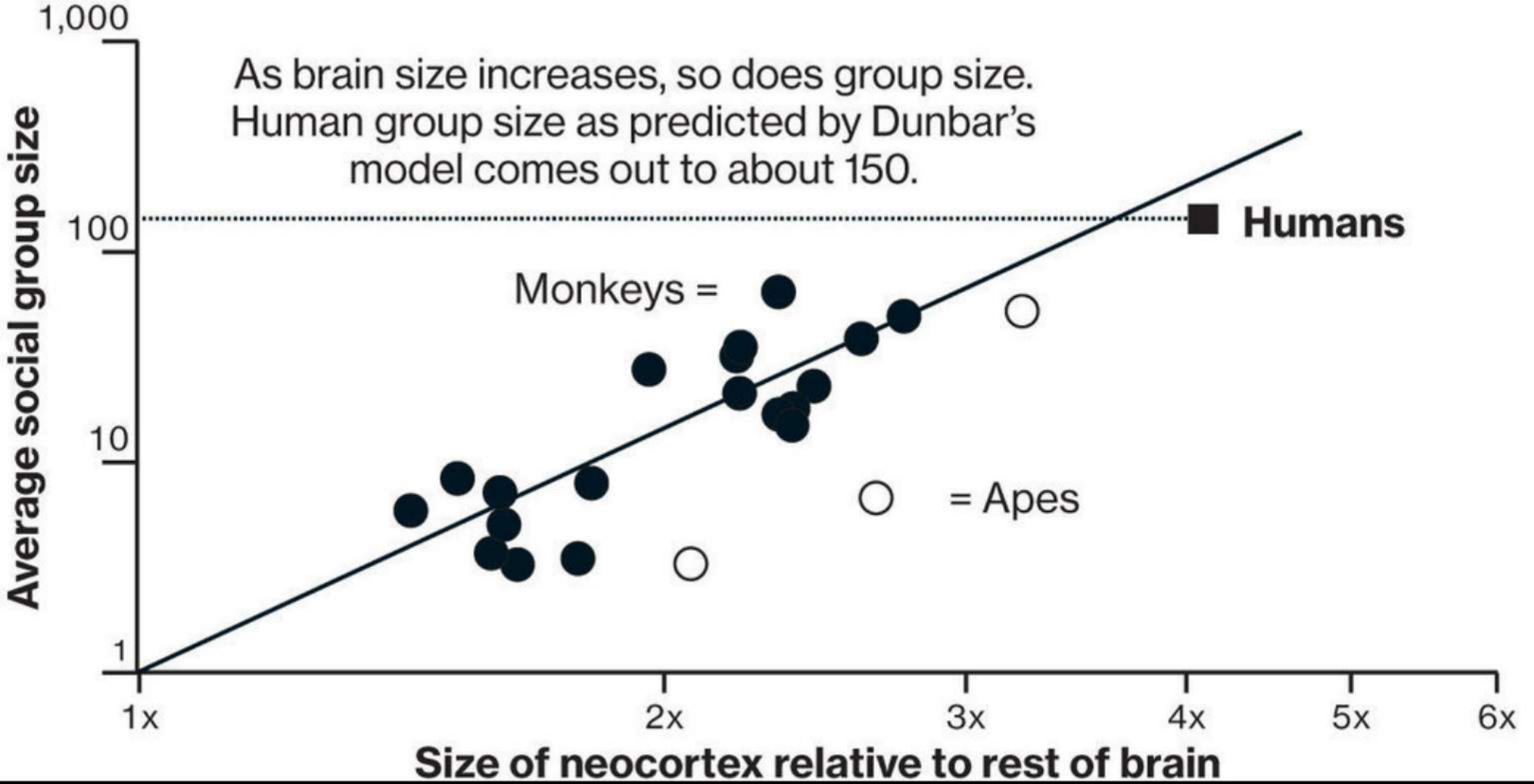


While SAFe is sold as "Scaling Agile to the Enterprise", the reality is that once we reach Dunbar's Number (around 150 people), you can no longer have effective PI planning meetings. There's just too many people to interact with effectively. This is a direct analogy to clock fanout problems on synchronous circuits where things like clock skew much be taken into account, and the root cause of why it's really tough to scale clocked CPUs past certain clock speeds.

Since SAFe requires commitment driven development to do all that planning with, and the errors in estimating software development time are way too variable to give good guidance in planning, the work committed to never matches the reality of the development process, and making the dates still requires heroes, death marches, and compromising on quality.






1,000
100
10
1
Average social group size
As brain size increases, so does group size.
Human group size as predicted by Dunbar's
model comes out to about 150.
Humans
Monkeys =
= Apes
1x
2x
3x
4x
5x
6x
Size of neocortex relative to rest of brain

DUNBAR'S NUMBER: 150

TYPICAL NUMBER OF PEOPLE WE CAN KEEP TRACK OF AND CONSIDER PART OF OUR ONGOING SOCIAL NETWORK

150 TRIBE

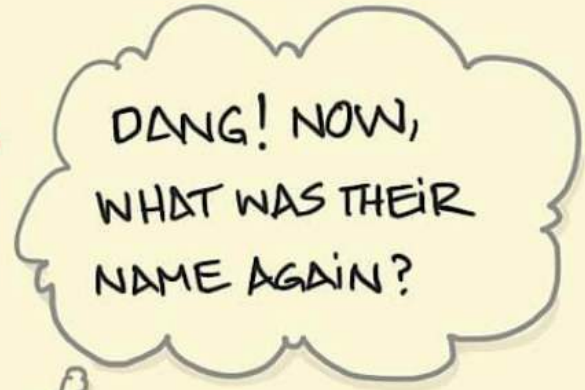
50 CLAN

15 SUPER FAMILY

5 CLOSE FRIENDS

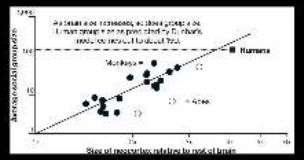
WEAKER TIES

MORE INVESTMENT IN RELATIONSHIP



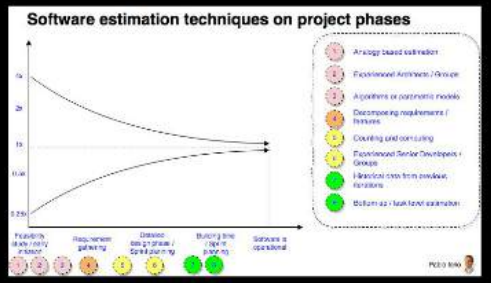
sketchplanations

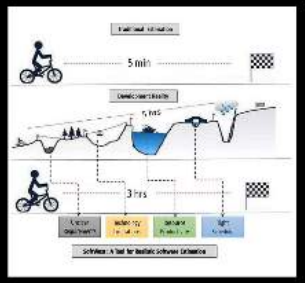
And Here's the two extremely damming aspects to all of this



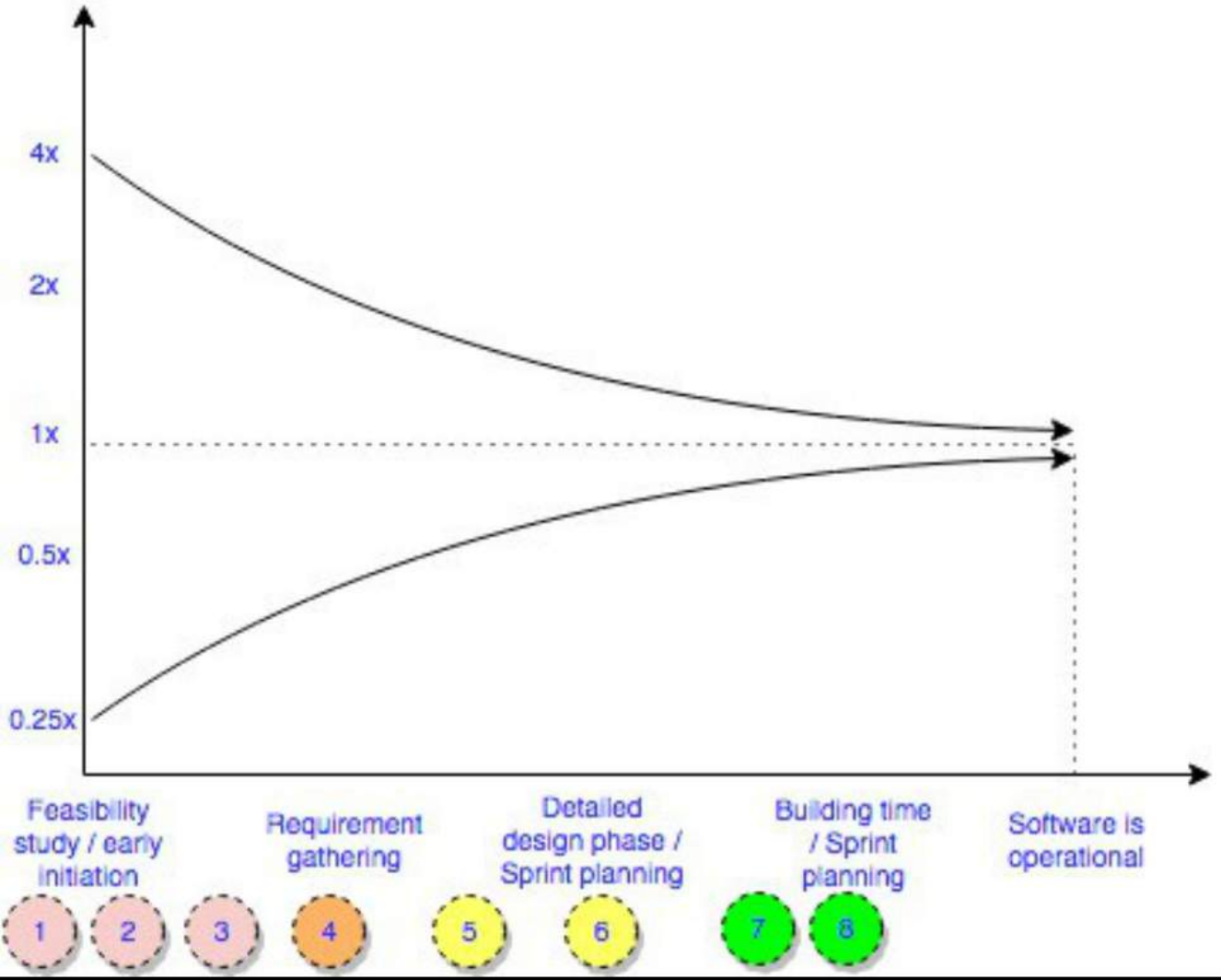
While SAFe is sold as "Scaling Agile to the Enterprise", the reality is that once we reach Dunbar's Number (around 150 people), you can no longer have effective PI planning meetings. There's just too many people to interact with effectively. This is a direct analogy to clock fanout problems on synchronous circuits where things like clock skew much be taken into account, and the root cause of why it's really tough to scale clocked CPUs past certain clock speeds.

Since SAFe requires commitment driven development to do all that planning with, and the errors in estimating software development time are way too variable to give good guidance in planning, the work committed to never matches the reality of the development process, and making the dates still requires heroes, death marches, and compromising on quality.



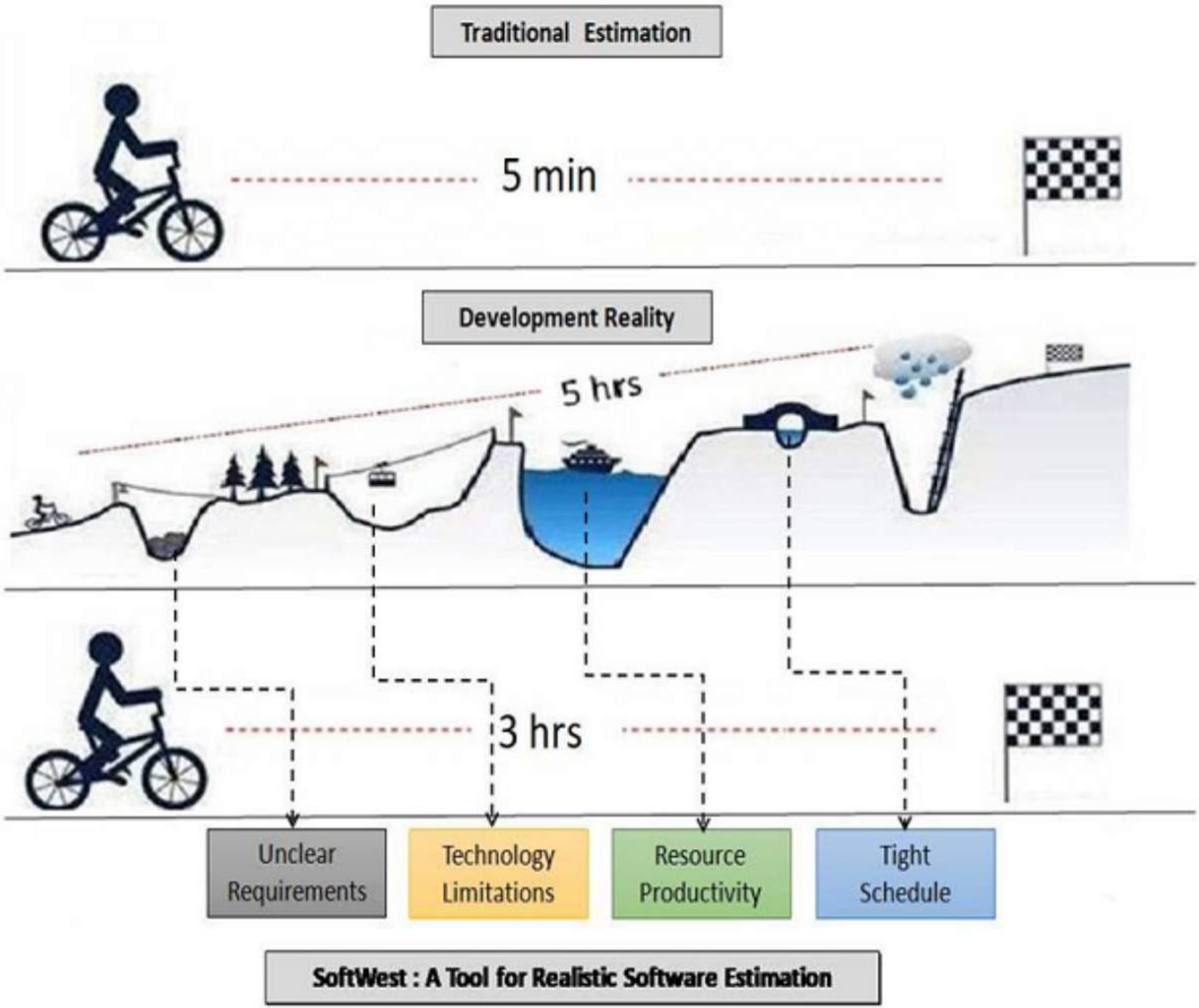


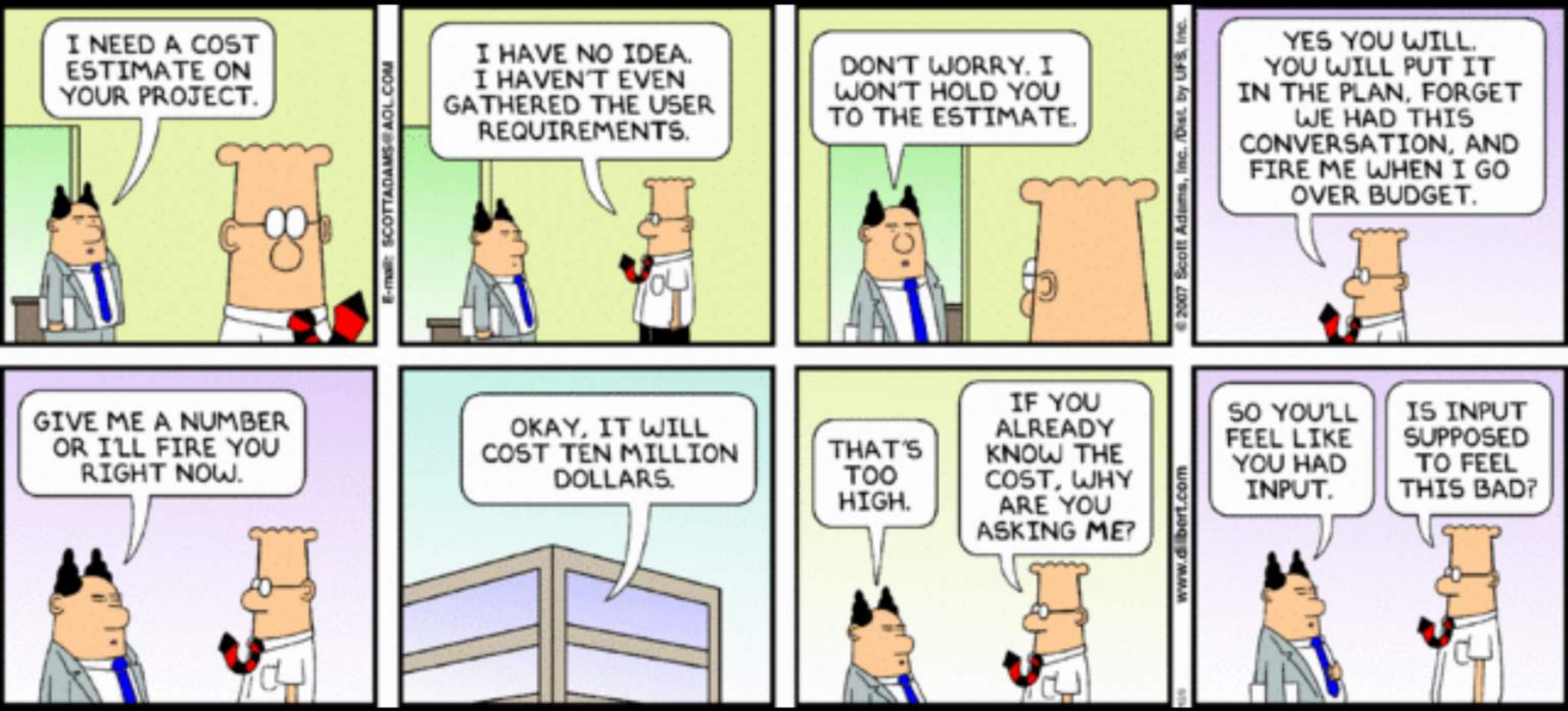
Software estimation techniques on project phases



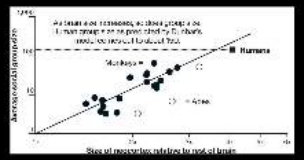
1. Analogy based estimation
2. Experienced Architects / Groups
3. Algorithms or parametric models
4. Decomposing requirements / features
5. Counting and computing
6. Experienced Senior Developers / Groups
7. Historical data from previous iterations
8. Bottom up / task level estimation

Pablo Iorio


Traditional Estimation
5 min
Development Reality
5 hrs
3 hrs
Unclear Requirements
Technology Limitations
Resource Productivity
Tight Schedule
SoftWest : A Tool for Realistic Software Estimation

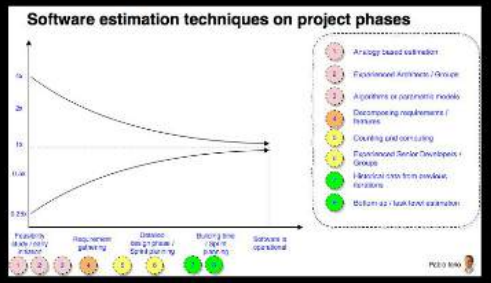

I NEED A COST ESTIMATE ON YOUR PROJECT.
I HAVE NO IDEA. I HAVEN'T EVEN GATHERED THE USER REQUIREMENTS.
DON'T WORRY. I WON'T HOLD YOU TO THE ESTIMATE.
YES YOU WILL. YOU WILL PUT IT IN THE PLAN, FORGET WE HAD THIS CONVERSATION, AND FIRE ME WHEN I GO OVER BUDGET.
GIVE ME A NUMBER OR I'LL FIRE YOU RIGHT NOW.
OKAY, IT WILL COST TEN MILLION DOLLARS.
THAT'S TOO HIGH.
IF YOU ALREADY KNOW THE COST, WHY ARE YOU ASKING ME?
SO YOU'LL FEEL LIKE YOU HAD INPUT.
IS INPUT SUPPOSED TO FEEL THIS BAD?
E-mail: SCOTTADAMS@AOL.COM
© 2007 Scott Adams, Inc. /Dist. by UFS, Inc.
www.dilbert.com

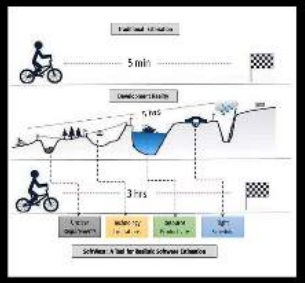
And Here's the two extremely damming aspects to all of this



While SAFe is sold as "Scaling Agile to the Enterprise", the reality is that once we reach Dunbar's Number (around 150 people), you can no longer have effective PI planning meetings. There's just too many people to interact with effectively. This is a direct analogy to clock fanout problems on synchronous circuits where things like clock skew much be taken into account, and the root cause of why it's really tough to scale clocked CPUs past certain clock speeds.

Since SAFe requires commitment driven development to do all that planning with, and the errors in estimating software development time are way too variable to give good guidance in planning, the work committed to never matches the reality of the development process, and making the dates still requires heroes, death marches, and compromising on quality.





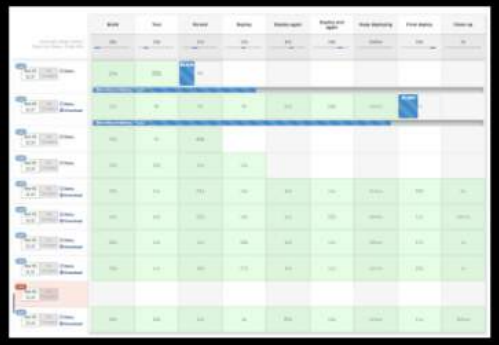
Here's the Real Problem to Solve

I want organizations to stop trying to scale up team Agile practices to work on the projects that are "Enterprise Big" and instead to work on scaling down those big monster monolithic systems into smaller loosely coupled systems of highly cohesive parts, and be smart about how the parts get wired together.

OK. Technically, one method of accomplishing this is to use microservices that work over versioned APIs that are tolerant of version mismatches, can automatically fall back to previous API versions, and anneal themselves by real time negotiation of versioning.

Yes. That will take software craftsmanship and major architectural refactoring for the majority of the monolithic systems that I encounter. [It's my plight that I'm never involved with the organizations that have these sorts of architectures in place already. I guess they have no need for me, since they are already doing the right things. :-)]

And What About ARTs, AKA CI Pipelines



This is nothing more than traditional DTAP (Development, Test, Accept, Production) thinking, with decision making, hand wringing, and wait states for approvals to occur.

Most of the Agile Release Train Wrecks occur when one DTAP pipeline does not pump sufficiently pure code for the integration pipelines to consume. The train's engines sputter and die horrible deaths instead of getting to where they needed to be.

You should always strive to develop full stack on trunk and deploy upon positive automated test results.

Releases should not be viewed as something abnormal and risky. Releases should happen all the time, because that's why we develop code in the first place.

Are You Saying That All the SAFe "Flow Model" Stuff is All Wrong?

Yeah. Much of it is wrong. But it's a good business!

I had to go to the doctor recently. Actually my gastroenterologist told me that he was out of ideas, and referred me to the "Mecca" for my condition. Mount Sinai Hospital in Manhattan.

I met a doctor there that was, bar none, the best doctor I've ever worked with. A rare combination of a very smart person who also has great bedside manner. He's actually a professor at the School of Medicine who also sees some patients.

And then I realized something about doctors. There are three types:

- Those who research, write papers, and present at conferences (like this new doctor).
- Those who read the occasional paper on topics they deal with (like my current doctor)
- Those who are licensed, but only learn of new things through pharmaceutical reps and product brochures.

And that's when I realized that it occurs in software development as well.

Let's Look for Clocks in This Popular "Scaled Agile Framework"

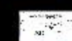
[illegible]

[illegible]

[illegible]

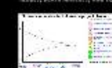
[illegible]

And Here's the two extremely damming aspects to all of this



While SAFe is sold as "Scaling Agile to the Enterprise", the reality is that once we reach Dunbar's Number (around 150 people), you can no longer have effective PI planning meetings. There's just too many people to interact with effectively. This is a direct analogy to clock fanout problems on synchronous circuits where things like clock skew must be taken into account, and the root cause of why it's really tough to scale clocked CPUs past certain clock speeds.

Since SAFe requires commitment before development to do all that planning with, and the errors in estimating software development time are way too variable to give good guidance in planning, the work committed to never matches the reality of the development process, and making the dates still requires heroes, death marches, and compromising on quality.



Here's the Real Problem to Solve

I want organizations to stop trying to scale up team Agile practices to work on the projects that are "Enterprise Big" and instead to work on scaling down those big monster monolithic systems into smaller loosely coupled systems of highly cohesive parts, and be smart about how the parts get wired together.

Ok. Technically, one method of accomplishing this is to use microservices that work over versioned APIs that are tolerant of version mismatches, can automatically fall back to previous API versions, and anneal themselves by real time negotiation of versioning.

Yes. That will take software craftsmanship and major architectural refactoring for the majority of the monolithic systems that I encounter. (It's my plight that I'm never involved with the organizations that have these sorts of architectures in place already. I guess they have no need for me, since they are already doing the right things. :-))

And What About ARTs, AKA CI Pipelines

This is nothing more than traditional DTAP (Development, Test, Accept, Production) thinking, with decision making, hand wringing, and wait states for approvals to occur.

Most of the Agile Release Train Wrecks occur when one DTAP pipeline does not pump sufficiently pure code for the integration pipelines to consume. The train's engines sputter and die horrible deaths instead of getting to where they needed to be.

You should always strive to develop full stack on trunk and deploy upon positive automated test results.

Releases should not be viewed as something abnormal and risky. Releases should happen all the time, because that's why we develop code in the first place.

WTF is this talk about?

A Tiny Bit of Theory About Digital Electronics - Synchronous vs Asynchronous

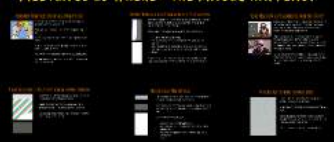
Famous Asynchronous Digital Logic Devices That You May Be Familiar With

Why Asynchronous is a Hot Topic in Digital Logic Design Today

How and Why are Organizations Becoming More Like Synchronous Digital Circuits?

Are You Saying That All the SAFe "Flow Model" Stuff is All Wrong?

Where did organizations get the idea that meetings is where "the magic happens?"



OK. Now, as an Organization, Let's Improve and Reduce Meetings. What Would That Look Like?

On Virtualization and Containerization and Why They Can Help

Closing Thoughts and Takeaways

Where did organizations get the idea that meetings is where "the magic happens?"

Remember Principle 6 in the Agile Manifesto?

"The most efficient and effective method of conveying information to and within a development team is face-to-face conversation."

Way too many teams and organizations that I could seem to have forgotten that one these days.

And I seem a lot with teams that love to scream "The Software sets our team development efforts!"

I find that these teams, aren't really teams – they're more like just a bunch of people that happen to be working on the same stuff.

Most of these teams have US based management and many have overseas based development.

So, why don't these teams get that cod-fashioned thing about face-to-face communication being the best communication?

Why Don't We Meet Face-to-Face? Some reasons off of the top of my head:

People work a lot from home. Or on some other location. Or in some other time zone. Or in some other language. Even though the default language is English these days, it's not the first language for the majority of developers in the world.

People are more comfortable working on a device, laptop or mobile, then just talking.

- Ahh. I don't wear in stores very much anymore.
- And one reason for that is that I'd prefer to not have to talk to anyone to just buy something.
- Look at your Smartphone. Did you know that you can actually call people with that. It rings them, they answer, and you have a realtime conversation? I know. Who would have thought?

We are told that all of our problems are solved with JIRA, Trackers, Rallydomo, Rally, Webex, Slack, Sococo, Teams and other "Enterprise Sanctioned Process and Collaboration Tools."

Do We Need to Be So Disconnected From Each Other?

By the very nature of the process tools, people on teams are isolated to work on the tool's interface. Alone and disconnected from one another. Perfecting their stories, tasks, and estimates. And, many times, recording actuals versus estimates. That doesn't sound very productive to me!

And by the very nature of the collaboration tools, people are still alone and disconnected from one another. Many times, fighting bridges, loss of connectivity, and saying a lot of "Can you hear me now?". Maybe that's a WebCam guy. And some people get to play with cool avatars or dress up their online image with a cool set of eyeglasses.

This leads to ceremonies, already just another clock tick., to become a recitation of what's on the process tool. Sort of a status of a status of a status.... And, of all the "Three questions of a stand-up", the least important one is "what I did yesterday."

It Seems That Ultimately, Tools Tempt Us to Do Too Much Work in Meetings

The ultimate result of these process and communications tools is to make our time together as the only time we get to act as a team.

And people tend to want to waste no work, because of a lack of trust by management and a lack of independent enablement by the organization.

They become a poor analogy of what would happen if the people on the team were actually interacting person-to-person.

Meetings Beget More Meetings

The more meetings that we tend to have results in follow-up meetings that have to be scheduled to further resolve and refine the work to be done.

From a Systems Thinking perspective, the fact that more meetings need to be scheduled results in yet more fragments of the day being in meetings in a destructive feedback loop.

From a Lean Wastes perspective, we lose in:

- Waiting on meetings, rather than completing work
- Transporting issues from one meeting to the next inevitably results in loss of important details
- Motion waste in the form of saying and re-establishing context from one meeting to the next
- Etc, etc.

Meetings Are Not Where the Magic Hapns

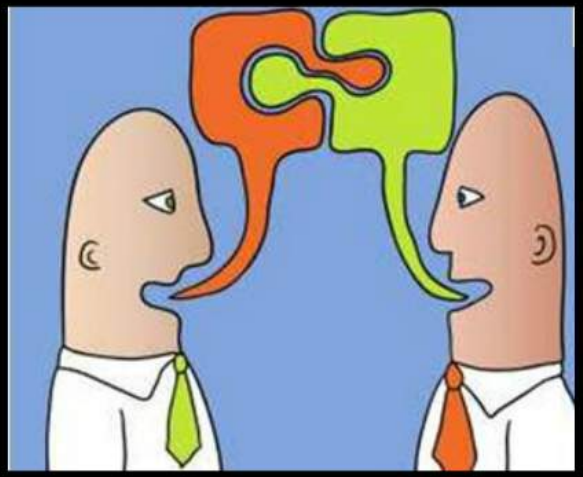
In fact, if there's one thing to take away from this talk, it's to recognize that meetings are just a Lean Waste.

No doubt some meetings are needed. But more meetings is just more Lean Waste.

Real team collaboration, where one developer leans over to another developer and asks to look at something, or ask for some help on getting something done is where the magic happens.

It's completely asynchronous, unscripted, and very productive!

Remember Principle 6 in the Agile Manifesto?



"The most efficient and effective method of conveying information to and within a development team is face-to-face conversation."

Way too many teams and organizations that I coach seem to have forgotten that one these days.

And I deal a lot with teams that love to tout how "The sun never sets on their development efforts!"

I find that these teams aren't really teams - they're more like just a bunch of people that happen to be working on the same stuff.

Most of these teams have US based management and many have overseas based development.

So, why don't these teams get that old fashioned thing about face-to-face communication being the best communication?

Why Don't We Meet Face-to-Face? Some reasons off of the top of my head:

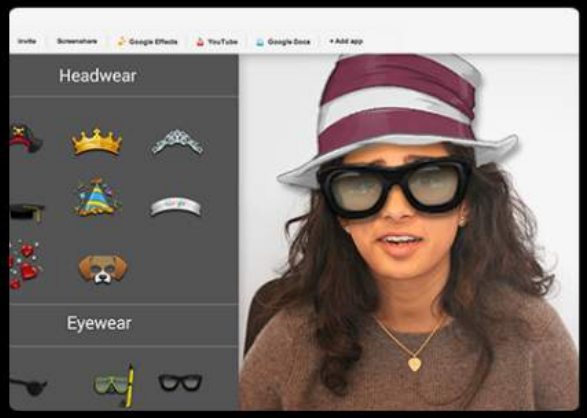
People work a lot from home. Or in some other location. Or in some other time zone. Or in some other language. Even though the default language is English these days, it's not the first language for the majority of developers in the world.

People are more comfortable working on a device, laptop or mobile, then just talking.

- Hey. I don't shop in stores very much anymore.
- And one reason for that is that I'd prefer to not have to talk to anyone to just buy something.
- Look at your Smartphone. Did you know that you can actually call people with this? It rings them, they answer, and you have a realtime conversation! I know. Who would have thought?

We are told that all of our problems are solved with JIRA, Tracker, VersionOne, Rally, WebEx, Slack, Sococo, Teams and other "Enterprise Industrialized Process and Collaboration Tools"

Do We Need to Be So Disconnected From Each Other?



By the very nature of the process tools, people on teams are isolated to work in the tool's interface. Alone and disconnected from one another. Perfecting their stories, tasks, and estimates. And, many times, recording actuals versus estimates. That doesn't sound very productive to me!

And by the very nature of the collaboration tools, people are still alone and disconnected from one another. Many times, fighting echoes, loss of connectivity, and saying a lot of "Can you hear me now?". Maybe there's a WebCam on. And maybe people get to play with cool avatars or dress up their online image with a cool set of eyeglasses.

This leads to ceremonies, already just another clock sync, to become a recitation of what's on the process tool. Sort of a status of a status of a status.... And, of all the "three questions of a stand-up", the least important one is "what I did yesterday."

It Seems That Ultimately, Tools Tempt Us to Do Too Much Work in Meetings

The ultimate result of these process and communications tools is to make our time together as the only time we get to act as a team.

And people tend to want to wait to work, because of a lack of trust by management and a lack of independent enablement by the organization.

They become a poor analogy of what would happen if the people on the team were actually interacting person-to-person.

PRYSM

BAD MEETING BINGO

Technology glitch brings meeting to a halt	Moderator is struggling with the video feed	Awkward silence while presenter finds file	Bad audio connection makes voices sound robotic	Can't get the WebEx to work
Whiteboard notes captured in unreadable smartphone photo	Conference call attendee "beeps" in late	Someone draws a venn diagram	Meeting attendee is double booked	Marathon meeting interrupted by a "bio break"
Attendee caught not paying attention	Mass confusion about the meeting agenda	FREE	Somebody didn't get the invite	Excessive presentation animation makes you dizzy
A healthy-minded individual brings a salad to a meeting	Someone asks the presenter to go back a slide	Someone steps out for a phone call	Someone kills time by asking about the weather	Attendee is doing other work during the meeting
Whiteboard notes from previous meeting have vanished	No one takes notes for the next steps/ action items	A task gets tabled—again	Someone is coughing or sniffling	Someone says, "Let's go back to the drawing board"

prysm.com/teamwork-transformed

It Seems That Ultimately, Tools Tempt Us to Do Too Much Work in Meetings

The ultimate result of these process and communications tools is to make our time together as the only time we get to act as a team.

And people tend to want to wait to work, because of a lack of trust by management and a lack of independent enablement by the organization.

They become a poor analogy of what would happen if the people on the team were actually interacting person-to-person.

Meetings Beget More Meetings

The short meetings that we tend to have result in follow-up meetings that have to be scheduled to further resolve and refine the work to be done.

From a Systems Thinking perspective, the fact that more meetings need to be scheduled results in yet more fragments of the day being in meetings in a destructive feedback loop.

From a Lean Wastes perspective, we lose in:

- Waiting on meetings, rather than completing work
- Transporting issues from each meeting to the next inevitably results in loss of important details
- Motion waste in the form of saving and re-establishing context from one meeting to the next
- Etc, etc.


WE'RE HAVING A MEETING TO DISCUSS EMPLOYEE RETENTION.
www.dilbert.com scottadams@aol.com
TELL THEM THAT EMPLOYEES QUIT BECAUSE THERE ARE TOO MANY USELESS MEETINGS.
WE WON'T BE GETTING INTO REASONS AT THE FIRST MEETING.


WHAT'S YOUR TAKE ON THIS, DILBERT?
Dilbert.com DilbertCartoonist@gmail.com
WHAT? SORRY. I WAS USING THIS TIME TO THINK ABOUT SOMETHING USEFUL.

"SO WE ARE ALL AGREED AT LAST! ... WE DEFER A FINAL DECISION UNTIL OUR NEXT MEETING....

Meetings Beget More Meetings

The short meetings that we tend to have result in follow-up meetings that have to be scheduled to further resolve and refine the work to be done.

From a Systems Thinking perspective, the fact that more meetings need to be scheduled results in yet more fragments of the day being in meetings in a destructive feedback loop.

From a Lean Wastes perspective, we lose in:

- Waiting on meetings, rather than completing work
- Transporting issues from each meeting to the next inevitably results in loss of important details
- Motion waste in the form of saving and re-establishing context from one meeting to the next
- Etc, etc.

Meetings Are Not Where the Magic Hapns

In fact, if there's one thing to take away from this talk, it's to recognize that meetings are just a Lean Waste.

No doubt some meetings are needed. But more meetings is just more Lean Waste.

Real time collaboration, where one developer leans over to another developer and asks to look at something, or ask for some help on getting something done is where the magic happens.

It's completely asynchronous, unscripted, and very productive!

Where did organizations get the idea that meetings is where "the magic happens?"

Remember Principle 6 in the Agile Manifesto?

"The most efficient and effective method of conveying information to and within a development team is face-to-face conversation."

Way too many teams and organizations that I could seem to have forgotten that one these days.

And I seem a lot with teams that love to scream "The Software sets our team development efforts!"

I find that these teams, aren't really teams – they're more like just a bunch of people that happen to be working on the same stuff.

Most of these teams have US based management and many have overseas based development.

So, why don't these teams get that cod-fashioned thing about face-to-face communication being the best communication?

Why Don't We Meet Face-to-Face? Some reasons off of the top of my head:

People work a lot from home. Or on some other location. Or in some other time zone. Or in some other language. Even though the default language is English these days, it's not the first language for the majority of developers in the world.

People are more comfortable working on a device, laptop or mobile, then just talking.

- Ahh. I don't wear in stores very much anymore.
- And one reason for that is that I'd prefer to not have to talk to anyone to just buy something.
- Look at your Smartphone. Did you know that you can actually call people with that. It rings them, they answer, and you have a realtime conversation? I know. Who would have thought?

We are told that all of our problems are solved with JIRA, Trackers, Rallydomo, Rally, Webex, Slack, Sococo, Teams and other "Enterprise Sanctioned Process and Collaboration Tools."

Do We Need to Be So Disconnected From Each Other?

By the very nature of the process tools, people on teams are isolated to work on the tool's interface. Alone and disconnected from one another. Perfecting their stories, tasks, and estimates. And, many times, recording actuals versus estimates. That doesn't sound very productive to me!

And by the very nature of the collaboration tools, people are still alone and disconnected from one another. Many times, fighting bridges, loss of connectivity, and saying a lot of "Can you hear me now?". Maybe that's a WebCam guy. And some people get to play with cool avatars or dress up their online image with a cool set of eyeglasses.

This leads to ceremonies, already just another clock tick., to become a recitation of what's on the process tool. Sort of a status of a status of a status.... And, of all the "Three questions of a stand-up", the least important one is "what I did yesterday."

It Seems That Ultimately, Tools Tempt Us to Do Too Much Work in Meetings

The ultimate result of these process and communications tools is to make our time together as the only time we get to act as a team.

And people tend to want to waste no work, because of a lack of trust by management and a lack of independent enablement by the organization.

They become a poor analogy of what would happen if the people on the team were actually interacting person-to-person.

Meetings Beget More Meetings

The more meetings that we tend to have results in follow-up meetings that have to be scheduled to further resolve and refine the work to be done.

From a Systems Thinking perspective, the fact that more meetings need to be scheduled results in yet more fragments of the day being in meetings in a destructive feedback loop.

From a Lean Wastes perspective, we lose in:

- Waiting on meetings, rather than completing work
- Transporting issues from one meeting to the next inevitably results in loss of important details
- Motion waste in the form of saying and re-establishing context from one meeting to the next
- Etc, etc.

Meetings Are Not Where the Magic Hapns

In fact, if there's one thing to take away from this talk, it's to recognize that meetings are just a Lean Waste.

No doubt some meetings are needed. But more meetings is just more Lean Waste.

Real team collaboration, where one developer leans over to another developer and asks to look at something, or ask for some help on getting something done is where the magic happens.

It's completely asynchronous, unscripted, and very productive!

WTF is this talk about?

A Tiny Bit of Theory About Digital Electronics - Synchronous vs Asynchronous

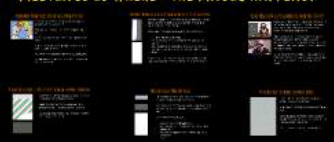
Famous Asynchronous Digital Logic Devices That You May Be Familiar With

Why Asynchronous is a Hot Topic in Digital Logic Design Today

How and Why are Organizations Becoming More Like Synchronous Digital Circuits?

Are You Saying That All the SAFe "Flow Model" Stuff is All Wrong?

Where did organizations get the idea that meetings is where "the magic happens?"



OK. Now, as an Organization, Let's Improve and Reduce Meetings. What Would That Look Like?

On Virtualization and Containerization and Why They Can Help

Closing Thoughts and Takeaways

OK. Now, as an Organization, Let's Improve and Reduce Meetings. What Would That Look Like?

Start With Retrospectives

For most "Agile/Scrum" teams I encounter, retrospectives are a waste of time doing a choreographed ceremony that has precious little value.

Try replacing that with good team conversations in real time, as something is learned (good or bad).

Like "What did we learn? What surprised you? And what will you do differently in the future?" if you need a place to start.

But establishing a feedback loop in an environment where people aren't afraid to start with something, they can say they don't know, have time to learn and figure it out, and then effectively share that with others is vital.

And another thing to try is to celebrate not just the successes we have (even though we really need them!), but also the failures. People need to not be afraid to think "outside the box", try something, and then try something else if it doesn't work out. Having a good time explaining what went wrong is probably even more important than rote application of something that worked before. It's how you make an environment where innovation happens!

And Then Start Single Streaming

Without having many non-value producing streams in motion, you will get a lot less Lean Wastes traceable to Transportation, Inventory, Motion, Waiting, and Defect, because you don't have to coordinate and plan as much, since you start to live in the here and now.

Stop starting so many things and tracking where they are each day and start finishing things, EVEN JUST ONE THING, each day as a team.

The ability to work as a real team is what makes it stronger than the sum of its parts.

Next, Burn Enterprise Scaling for Agility to the Ground

Stop trying to scale Agile with all of the planning, preplanning, post planning, SWAG estimation, high accuracy estimation, etc.

Stop obsessing about the cost of development or the cost accounting of it. Let people like ScrumMasters handle those dirty tasks if you really need to. Learn to track value added, not cost as measured by hours worked.

Replace that planning with autonomy of T-shaped full-stack teams that work directly for the business owner.

Hire good people and trust them to make good choices for which customers to please, with what, and when.

And Don't Forget to Practice Good Engineering!

Embrace the DevOps Culture and a good architecture that permits trunk development and Continuous Delivery.

Remember that when you do software development, you're involved in Product Development, not Project Delivery.

So, software is never really ever done. You want it to work and then improve it over time.

Make sure that your engineering practices and architecture reflect that mindset.

Start With Retrospectives

For most "Agile/Scrum" teams I encounter, retrospectives are a waste of time doing a choreographed ceremony that has precious little value.

Try Replacing that with good team interactions in realtime, as something is learned (good or bad).

I like "What did we learn? What surprised you? And what will you do differently in the future" if you need a place to start.

But establishing a feedback loop in an environment where people aren't afraid to start with something they can say they don't know, have time to learn and figure it out, and then effectively share that with others is vital.

And another thing to try is to celebrate not just the successes we have (even though we really need them!), but also the failures. People need to not be afraid to think "outside the box", try something, and then try something else if it doesn't work out. Having a good time explaining what went wrong is probably even more important than mere application of something that worked before. It's how you make an environment where innovation happens!

And Then Start Single Streaming

Without having many non-value producing streams in motion, you will get a lot less Lean Wastes traceable to Transportation, Inventory, Motion, Waiting, and Defect, because you don't have to coordinate and plan as much, since you start to live in the here and now.

Stop starting so many things and tracking where they are each day and start finishing things, EVEN JUST ONE THING, each day as a team.

The ability to work as a real team is what makes it stronger than the sum of its parts.

Next, Burn Enterprise Scaling for Agility to the Ground

stop trying to scale Agile with all of the planning, preplanning, post planning, SWAG estimation, high accuracy estimation, etc.

Stop obsessing about the cost of development or the cost accounting of it. Let people like ScrumMasters handle those dirty tasks if you really need to. Learn to track value added, not cost as measured by hours worked.

Replace that planning with autonomy of T-shaped full-stack teams that work directly for the business owner.

Hire good people and trust them to make good choices for which customers to please, with what, and when.

And Don't Forget to Practice Good Engineering!

Embrace the DevOps Culture and a good architecture that permits trunk development and Continuous Delivery.

Remember that when you do software development, you're involved in Product Development, not Project Delivery.

So, software is never really ever done. You want it to work and then improve it over time.

Make sure that your engineering practices and architecture reflect that mindset.

OK. Now, as an Organization, Let's Improve and Reduce Meetings. What Would That Look Like?

Start With Retrospectives

For most "Agile/Scrum" teams I encounter, retrospectives are a waste of time doing a choreographed ceremony that has precious little value.

Try replacing that with good team conversations in real time, as something is learned (good or bad).

Like "What did we learn? What surprised you? And what will you do differently in the future?" if you need a place to start.

But establishing a feedback loop in an environment where people aren't afraid to start with something they can say they don't know, have time to learn and figure it out, and then effectively share that with others is vital.

And another thing to try is to celebrate not just the successes we have (even though we really need them), but also the failures. People need to not be afraid to think "outside the box", try something, and then try something else if it doesn't work out. Having a good time explaining what went wrong is probably even more important than mere application of something that worked before. It's how you make an environment where innovation happens!

And Then Start Single Streaming

Without having many non-value producing streams in motion, you will get a lot less Lean Wastes traceable to Transportation, Inventory, Motion, Waiting, and Defect, because you don't have to coordinate and plan as much, since you start to live in the here and now.

Stop starting so many things and tracking where they are each day and start finishing things, EVEN JUST ONE THING, each day as a team.

The ability to work as a real team is what makes it stronger than the sum of its parts.

Next, Burn Enterprise Scaling for Agility to the Ground

Stop trying to scale Agile with all of the planning, preplanning, post planning, SWAG estimation, high accuracy estimation, etc.

Stop obsessing about the cost of development or the cost accounting of it. Let people like ScrumMasters handle those dirty tasks if you really need to. Learn to track value added, not cost as measured by hours worked.

Replace that planning with autonomy of T-shaped full-stack teams that work directly for the business owner.

Hire good people and trust them to make good choices for which customers to please, with what, and when.

And Don't Forget to Practice Good Engineering!

Embrace the DevOps Culture and a good architecture that permits trunk developments and Continuous Delivery.

Remember that when you do software development, you're involved in Product Development, not Project Delivery.

So, software is never really ever done. You want it to work and then improve it over time.

Make sure that your engineering practices and architecture reflect that mindset.

WTF is this talk about?

A Tiny Bit of Theory About Digital Electronics - Synchronous vs Asynchronous

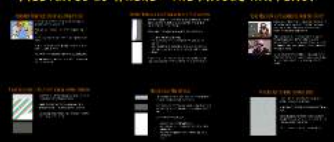
Famous Asynchronous Digital Logic Devices That You May Be Familiar With

Why Asynchronous is a Hot Topic in Digital Logic Design Today

How and Why are Organizations Becoming More Like Synchronous Digital Circuits?

Are You Saying That All the SAFe "Flow Model" Stuff is All Wrong?

Where did organizations get the idea that meetings is where "the magic happens?"



OK. Now, as an Organization, Let's Improve and Reduce Meetings. What Would That Look Like?

On Virtualization and Containerization and Why They Can Help

Closing Thoughts and Takeaways

On Virtualization and Containerization and Why They Can Help

The Goal is Shorter Concept to Cash, Not Great Schedules!

One thing that is many times overlooked as organizations try to get to shorter "concept to cash" cycle times is the role of Infrastructure As Code and Containerization tools.

Old style waterfall predictive model thinking was that driven. We wanted to have a smooth transition to production, so we started planning as early as possible, doing things like scheduling things like "QA" systems (managed servers), and working grand fine grained plans as early as possible.

But that old adage from Mary and Tom Poppendieck of "Delay decisions until the last responsible moment, when you have the most information, and can make the best decisions" has a load of relevance here.

Software development and integration testing does not work from a schedule the way that we wish it would.

We Want to Pull Solutions, Not Push Schedules

Don't start thinking requires extensive precise estimate planning to form a "push schedule" by way of the timelines on the teams involved.

A more Agile approach is to allow a "pull solution", where once we know that we need a test system of some form, we use the principles of Infrastructure as Code and create the system from unmanaged Cloud resources "just in time."

This solution has many advantages:

- There is less likelihood that things will go wrong than where we have a centralization process that results in the promotion of some test system into the production system, because we have practiced building the system many times before production (of course, with different instance capacities for each environment)
- It's much more in line with the DevOps concept of "cattle versus pets".
- We divorced ourselves from yet another synchronous clock cycle that slowed us down before.

Why Containers Make So Much Sense

In keeping with the "pull" approach, the use of Containers for housing developed modules extends and enhances. Now:

- We can test on the desktop, CI server, and all upstream environments in the very same fashion that we will use for production. Earlier testing is always better testing.
- We can make it real easy and very effective to network these systems together for integration using tools like Docker-Compose to make networking and data storage easy for when we don't need to worry about those concerns.
- Deploying components into Containers and then deploying Containers to create systems enhances creation of reproducible artifacts that can be built early on, tested often, and then reliably deployed with dependencies encapsulated and pre-verified before final deployment into production.

And, Oh By The Way, While You Transition Your Architecture to a Better Place...

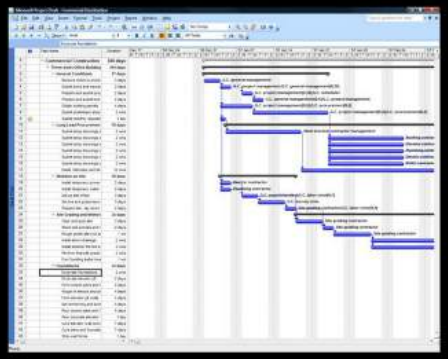
During the transition to fine grained micro services, break dependencies on components you are waiting on and place fresh on easily deployed containers.

Also make sure that you deploy test doubles (stacks) at the earliest possible time for all deployed upon components to permit early integration on executable requirements under the defined acceptance criteria of development.

That way, there is no waiting on completion for early integration. We can be asynchronous and end up in a better place while we're at it.

That is, move testing as far left as possible.

The Goal is Shorter Concept to Cash, Not Great Schedules!



One thing that is many times overlooked as organizations try to get to shorter "concept to cash" cycle times is the role of Infrastructure as Code and Containerization tools.

Old style waterfall predictive model thinking was DTAP driven. We wanted to have a smooth transition to Production, so we started planning as early as possible, doing things like scheduling things like "QA" systems (managed servers), and making grand fine grained plans as early as possible.

But that old adage from Mary and Tom Poppendieck of "Delay decisions until the last responsible moment, when you have the most information, and can make the best decisions" has a load of relevance here.

Software development and integration testing does not work from a schedule the way that we wish it would.

We Want to Pull Solutions, Not Push Schedules

That DTAP thinking requires extensive precise estimate planning to form a "push schedule" by way of the timelines on the teams involved.

A more Agile approach is to allow a "pull solution", where once we know that we need a test system of some form, we use the principles of Infrastructure as Code and create the system from unmanaged Cloud resources "just in time."

This solution has many advantages:

- There is less likelihood that things will go wrong than where we have a certification process that results in the promotion of some test system into the production system, because we have practiced building the system many times before production (of course, with different instance capacities for each environment).
- It's much more in line with the DevOps concept of "cattle versus pets".
- We divorced ourselves from yet another synchronous clock cycle that slowed us down before.

Why Containers Make So Much Sense

In keeping with the "pull" approach, the use of Containers for housing developed modules extends and enhances. Now:

- We can test on the desktop, CI server, and all upstream environments in the very same fashion that we will use for production. Earlier testing is always better testing.
- We can make it real easy and very effective to network these systems together for integration using tools like docker-compose to make networking and data storage easy for when we don't need to worry about those concerns.
- Deploying components into containers and then deploying containers to create systems enhances creation of reproducible artifacts that can be built early on, tested often, and then reliably deployed with dependencies encapsulated and pre-verified before final deployment into production.

And, Oh By The Way, While You Transition Your Architecture to a Better Place...

During the transition to fine grained micro services, break dependencies on components you are waiting on and place them in easily deployed containers.

Also make sure that you deploy test doubles (fakes) at the earliest possible time for all depended upon components to permit early integration on executable requirements which we defined at beginning of development.

That way, there is no waiting on completion for early integration. We can be asynchronous and end up in a better place while we're at it.

That is, move testing as far left as possible.

On Virtualization and Containerization and Why They Can Help

The Goal is Shorter Concept to Cash, Not Great Schedules!

One thing that is many times overlooked as organizations try to get to shorter "concept to cash" cycle times is the role of Infrastructure As Code and Containerization tools.

Old style waterfall predictive model thinking was that driven. We wanted to have a smooth transition to production, so we started planning as early as possible, doing things like scheduling things like "QA" systems (managed servers), and working grand fine grained plans as early as possible.

But that old adage from Mary and Tom Poppendieck of "Delay decisions until the last responsible moment, when you have the most information, and can make the best decisions" has a load of relevance here.

Software development and integration testing does not work from a schedule the way that we wish it would.

We Want to Pull Solutions, Not Push Schedules

Don't start thinking requires extensive precise estimate planning to form a "push schedule" by way of the timelines on the teams involved.

A more Agile approach is to allow a "pull solution", where once we know that we need a test system of some form, we use the principles of Infrastructure as Code and create the system from unmanaged Cloud resources "just in time."

This solution has many advantages:

- There is less likelihood that things will go wrong than where we have a centralization process that results in the promotion of some test system into the production system, because we have practiced building the system many times before production (of course, with different instance capacities for each environment)
- It's much more in line with the DevOps concept of "cattle versus pets".
- We divorced ourselves from yet another synchronous clock cycle that slowed us down before.

Why Containers Make So Much Sense

In keeping with the "pull" approach, the use of Containers for housing developed modules extends and enhances. Now:

- We can test on the desktop, CI server, and all upstream environments in the very same fashion that we will use for production. Earlier testing is always better testing.
- We can make it real easy and very effective to network these systems together for integration using tools like Docker-Compose to make networking and data storage easy for when we don't need to worry about those concerns.
- Deploying components into Containers and then deploying Containers to create systems enhances creation of reproducible artifacts that can be built early on, tested often, and then reliably deployed with dependencies encapsulated and pre-verified before final deployment into production.

And, Oh By The Way, While You Transition Your Architecture to a Better Place...

During the transition to fine grained micro services, break dependencies on components you are waiting on and place fresh on easily deployed containers.

Also make sure that you deploy test doubles (stacks) at the earliest possible time for all deployed upon components to permit early integration on executable requirements under the defined acceptance criteria of development.

That way, there is no waiting on completion for early integration. We can be asynchronous and end up in a better place while we're at it.

That is, move testing as far left as possible.

WTF is this talk about?

A Tiny Bit of Theory About Digital Electronics - Synchronous vs Asynchronous

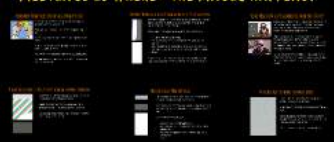
Famous Asynchronous Digital Logic Devices That You May Be Familiar With

Why Asynchronous is a Hot Topic in Digital Logic Design Today

How and Why are Organizations Becoming More Like Synchronous Digital Circuits?

Are You Saying That All the SAFe "Flow Model" Stuff is All Wrong?

Where did organizations get the idea that meetings is where "the magic happens?"



OK. Now, as an Organization, Let's Improve and Reduce Meetings. What Would That Look Like?

On Virtualization and Containerization and Why They Can Help

Closing Thoughts and Takeaways

Closing Thoughts and Takeaways

There's a difference between Synchronous and Asynchronous Digital Circuits.

And it has to do with a clock.

Synchronous circuits are not the most efficient nor effective circuits that we can make.

But they are the easiest.

So is it with organizations.

Organizations use clock signals just like synchronous circuits.

We just aren't trained to see them or to think that they could be a bad thing.

There are a lot of vendors out there who want to sell you processes and software that will fix your organization's problems.

But most of those products ultimately are sort of like putting lipstick on a pig and giving them an Apple Watch.

You're going to get dirty, and the pig is not going to like it.

You aren't going to get rid of the synchronous nature of your organization by next Monday and make it run as fast as the fastest Asynchronous CPUs that amaze us.

But there are things that we can iteratively try that can help.

Understanding that you have a problem is the first step.

Keep looking for that water and think about the implications of swimming in it like Agile Fish.

So Long,
and Thanks for
All the Fish

Questions.
Answers.
Debate.
Depart.

There's a difference between Synchronous and Asynchronous Digital Circuits.

And it has to do with a clock.

Synchronous circuits are not the most efficient nor effective circuits that we can make.

But they are the easiest.

So is it with organizations.

Organizations use clock signals just like synchronous circuits.

We just aren't trained to see them or to think that they could be a bad thing.

There are a lot of vendors out there who want to sell you processes and software that will fix your organization's problems.

But most of those products ultimately are sort of like putting lipstick on a pig and giving them an Apple Watch.

You're going to get dirty, and the pig is not going to like it.

You aren't going to get rid of the synchronous nature of your organization by next Monday and make it run as fast as the fastest Asynchronous CPUs that amaze us.

But there are things that we can iteratively try that can help.

Understanding that you have a problem is the first step.

Keep looking for that water and think about the implications of swimming in it like Agile Fish.

So Long,
and Thanks for
All the Fish

Questions.

Answers.

Debate.

Depart.

Closing Thoughts and Takeaways

There's a difference between Synchronous and Asynchronous Digital Circuits.

And it has to do with a clock.

Synchronous circuits are not the most efficient nor effective circuits that we can make.

But they are the easiest.

So is it with organizations.

Organizations use clock signals just like synchronous circuits.

We just aren't trained to see them or to think that they could be a bad thing.

There are a lot of vendors out there who want to sell you processes and software that will fix your organization's problems.

But most of those products ultimately are sort of like putting lipstick on a pig and giving them an Apple Watch.

You're going to get dirty, and the pig is not going to like it.

You aren't going to get rid of the synchronous nature of your organization by next Monday and make it run as fast as the fastest Asynchronous CPUs that amaze us.

But there are things that we can iteratively try that can help.

Understanding that you have a problem is the first step.

Keep looking for that water and think about the implications of swimming in it like Agile Fish.

So Long,
and Thanks for
All the Fish

Questions.
Answers.
Debate.
Depart.

WTF is this talk about?

A Tiny Bit of Theory About Digital Electronics - Synchronous vs Asynchronous

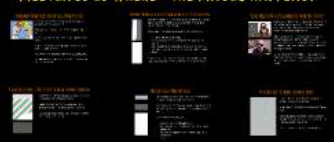
Famous Asynchronous Digital Logic Devices That You May Be Familiar With

Why Asynchronous is a Hot Topic in Digital Logic Design Today

How and Why are Organizations Becoming More Like Synchronous Digital Circuits?

Are You Saying That All the SAFe "Flow Model" Stuff is All Wrong?

Where did organizations get the idea that meetings is where "the magic happens?"



OK. Now, as an Organization, Let's Improve and Reduce Meetings. What Would That Look Like?

On Virtualization and Containerization and Why They Can Help

Closing Thoughts and Takeaways

Synchronous vs Asynchronous Digital Circuits as an Analogy to Organizational Dysfunction and Applied to DevOps Practices