From Program to Mixed SW/HW Implementation: How to Get It Right

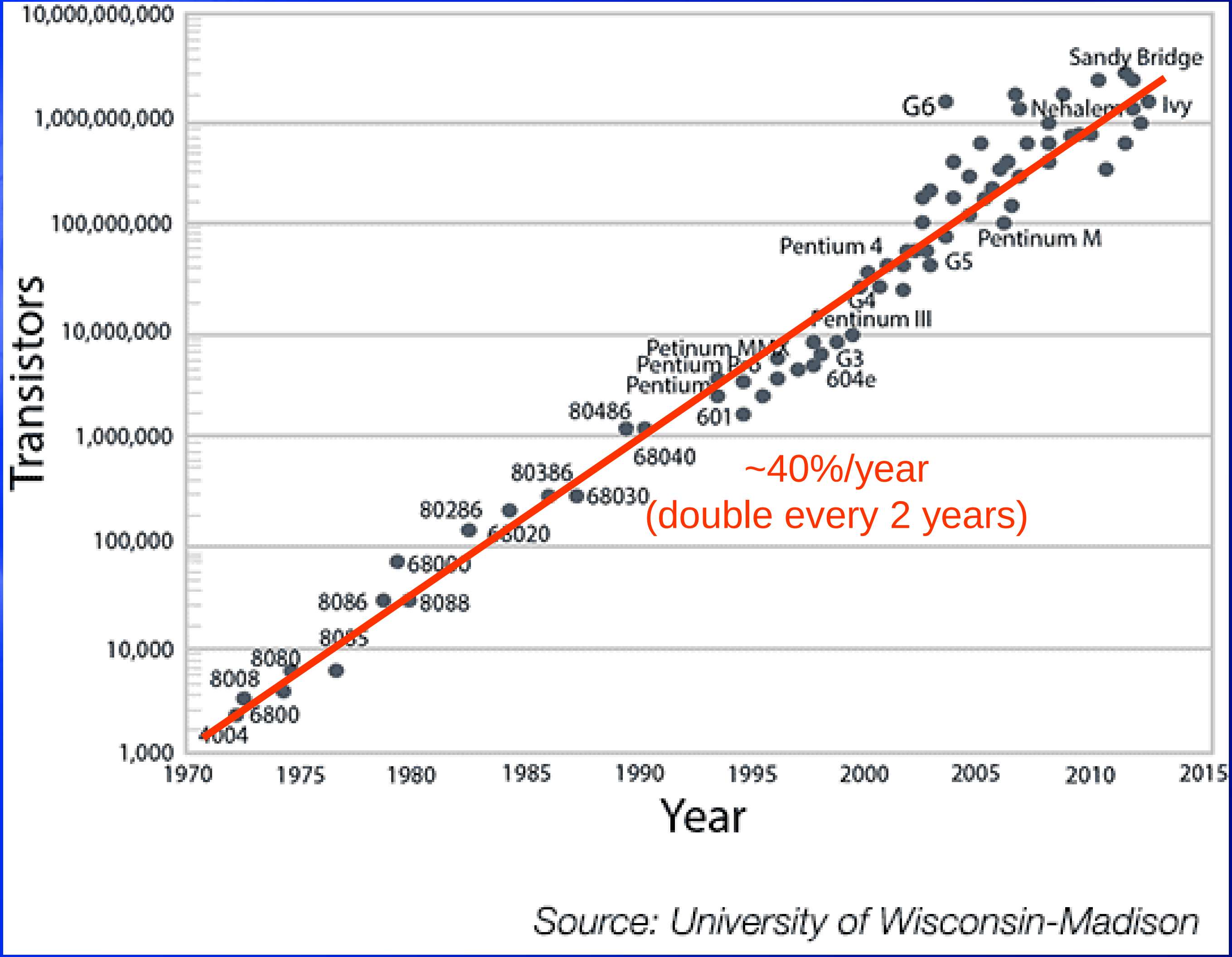
Carl Seger

Feb. 17, 2017

Outline

- Motivation
- Problem statement
- Previous work
- Suggested solution
- Research Questions

Moore's Law


10,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
Transistors
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
Year
4004
8008
8080
6800
8085
8086
8088
68000
80286
68020
80386
68030
68040
80486
Pentium
Pentium Pro
Petinum MMX
601
604e
G3
Pentium III
G4
Pentium 4
G5
Pentinum M
G6
Nehalem
Sandy Bridge
Ivy
~40%/year
(double every 2 years)
Source: University of Wisconsin-Madison

Power Wall

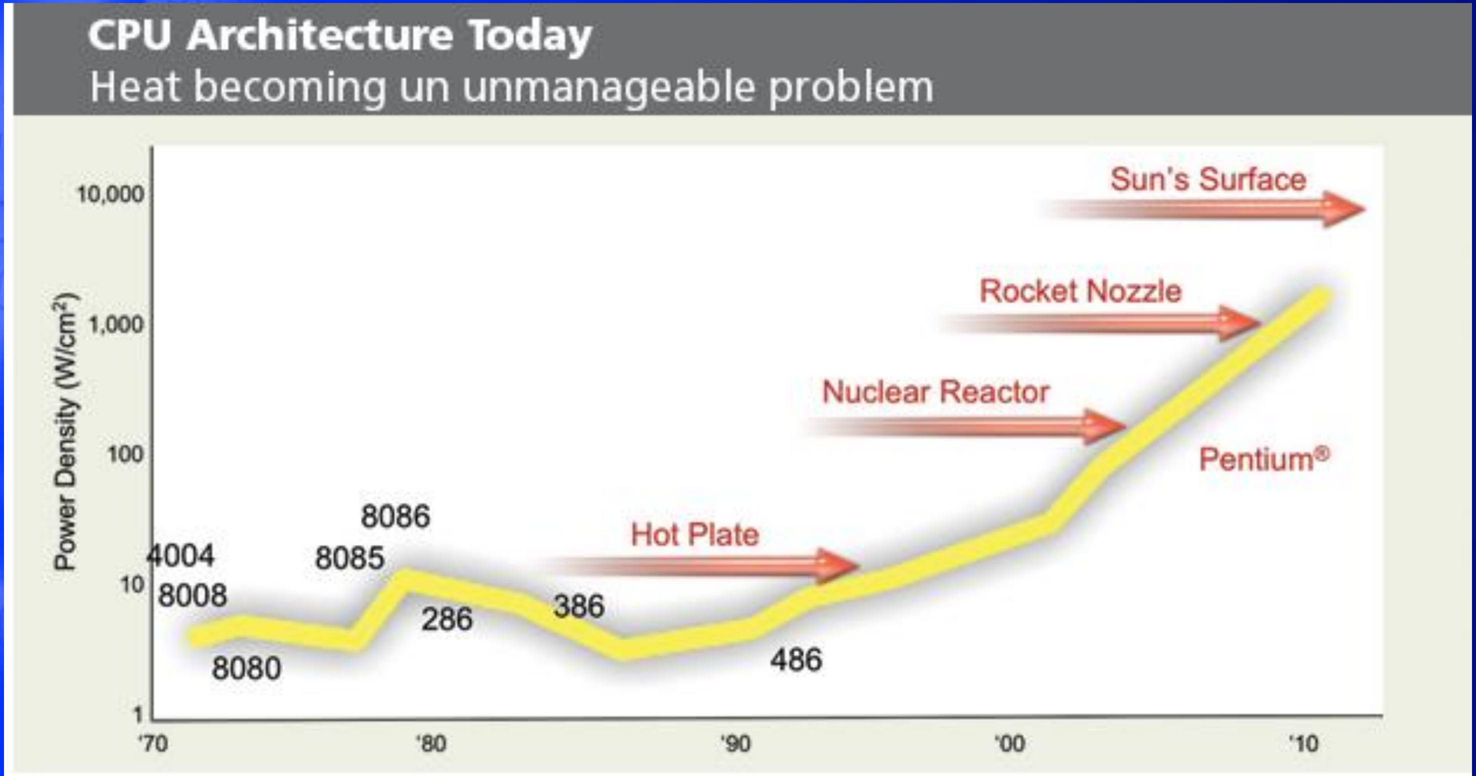
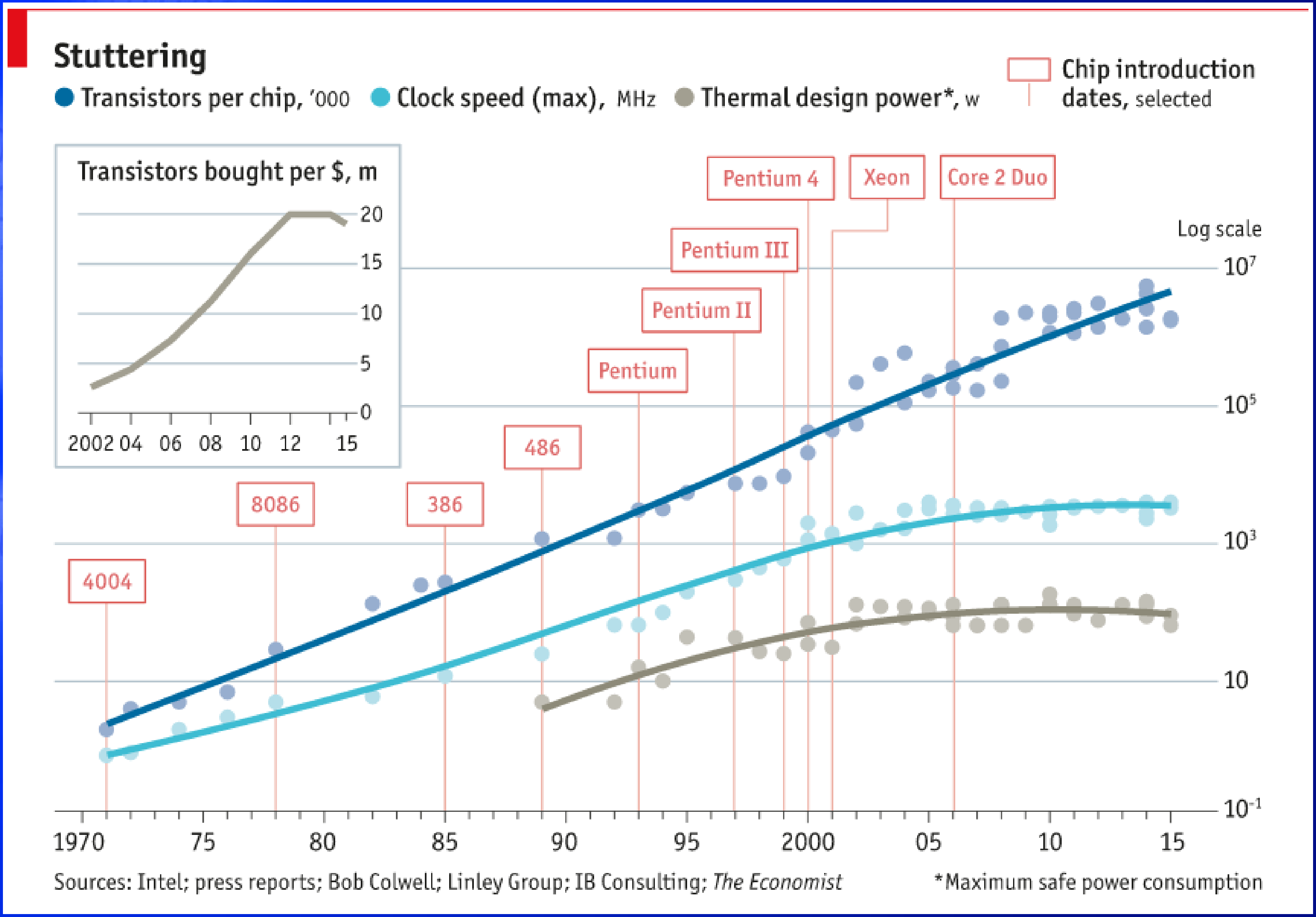


Figure 1. In CPU architecture today, heat is becoming an unmanageable problem. (Courtesy of Pat Gelsinger, Intel Developer Forum, Spring 2004)

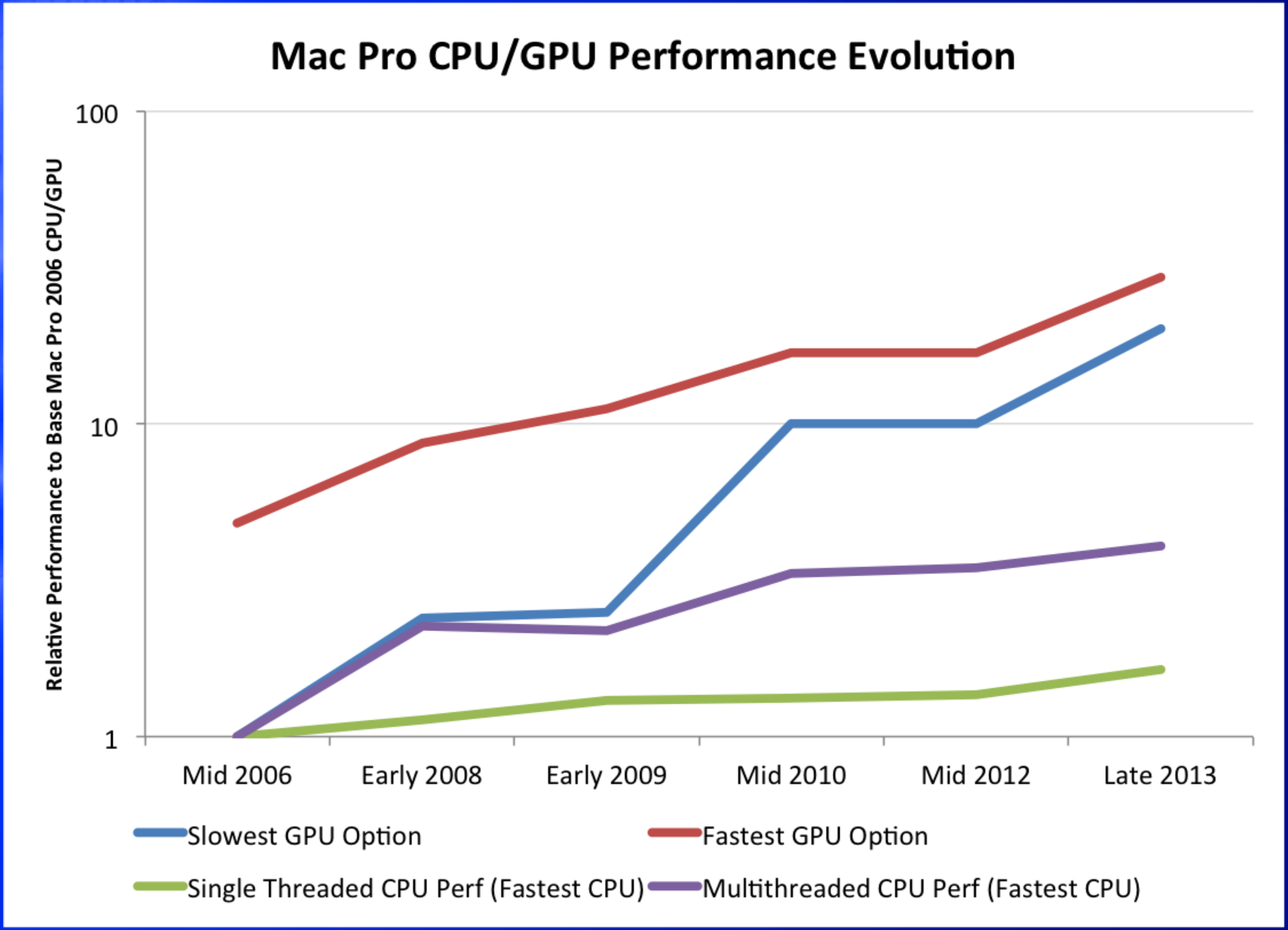
As a Result



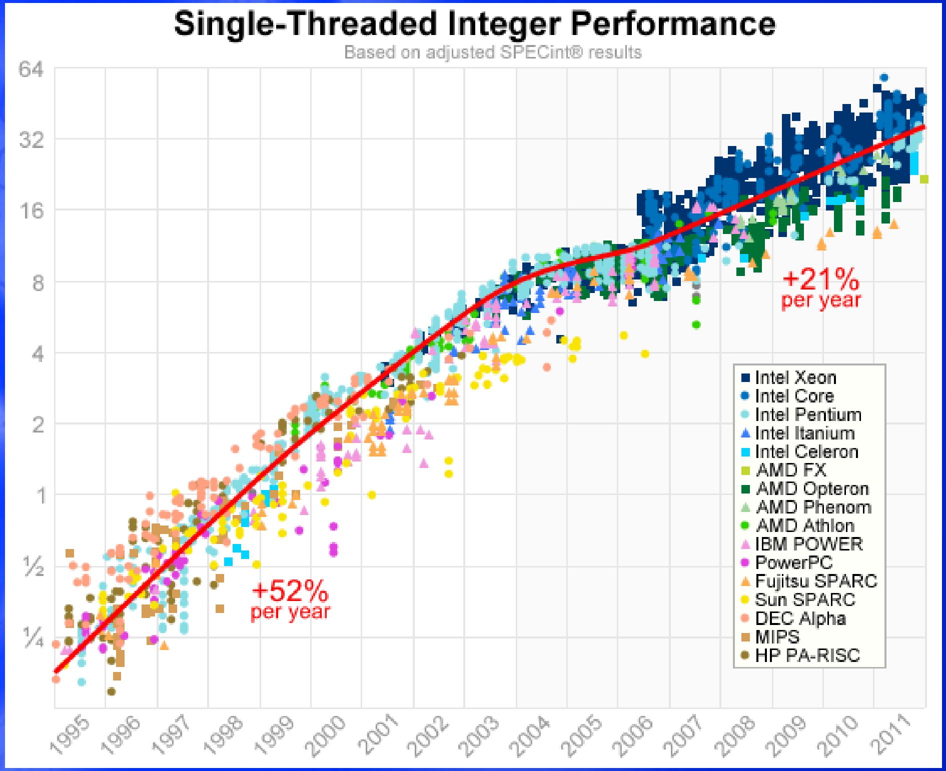
Sources: Intel; press reports; Bob Colwell; Linley Group; IB Consulting; *The Economist*

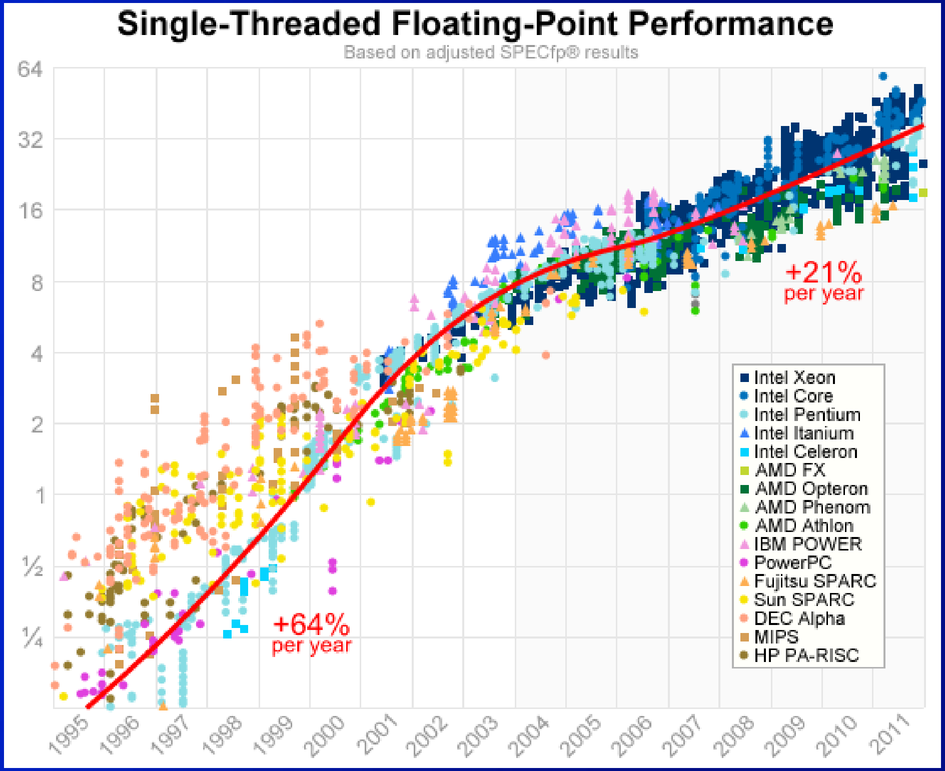
*Maximum safe power consumption

In Practice


Mac Pro CPU/GPU Performance Evolution
Relative Performance to Base Mac Pro 2006 CPU/GPU
100
10
1
Mid 2006
Early 2008
Early 2009
Mid 2010
Mid 2012
Late 2013
Slowest GPU Option
Fastest GPU Option
Single Threaded CPU Perf (Fastest CPU)
Multithreaded CPU Perf (Fastest CPU)

Single-Treaded Performance





Who Cares: Just Parallelize…

- Theoretical problem:
  - If NC != P, then there are <u>problems</u> that cannot be parallelized efficiently!
- Practical problem:
  - Only known <u>algorithms</u> are inherently sequential
    - The choice of computation depends critically (and immediately) on the result just computed.
- Even more practical problem:
  - A lot of useful and practical algorithms are highly sequential but need to be sped up!

NC = Nick's class; problems solvable on $O(n^k)$ machines in $O((\log(n))^c)$ time.

How to Increase ST-Performance

- Higher clock frequency
  - None: power wall has stopped this…
- Higher instructions-per-cycle (IPC)
  - Marginal: architects have pretty much wrung out most of it
- Better branch-prediction
  - Marginal: modern branch-predictors are about as good as it gets
- More advanced compilers
  - Marginal: “It’s the actual data, stupid”
- Off-load work onto (specialized) hardware
  - Potentially huge impact (2-3 orders of magnitude speed up)

Hardware Acceleration

- Alternatives:
  - New instructions in CPU
    - E.g., The AES-class instructions in x86 architecture
  - Specialized HW support in CPU (“custom CPU”)
    - E.g., Intel builds special CPUs for Facebook/Microsoft/…
  - Custom chip
    - GPUs, video codecs in chipsets
  - Field Programmable Gate Array (FPGA)
    - Traditionally on the PCIe bus, but now integrated with CPU

HW Acceleration cont.

- By 2020, it is projected that:
  - Every person will create ~1.5Gbyte of data per day
  - An autonomous car will create ~40Gbyte of data per hour
  - 3-D sports casts will create 2000Gbyte of data per minute
- It is clear that HW acceleration is critical!
- At the same time, the algorithms used are changing and improving rapidly.
  - Fixed HW is unlikely to keep up

Today:

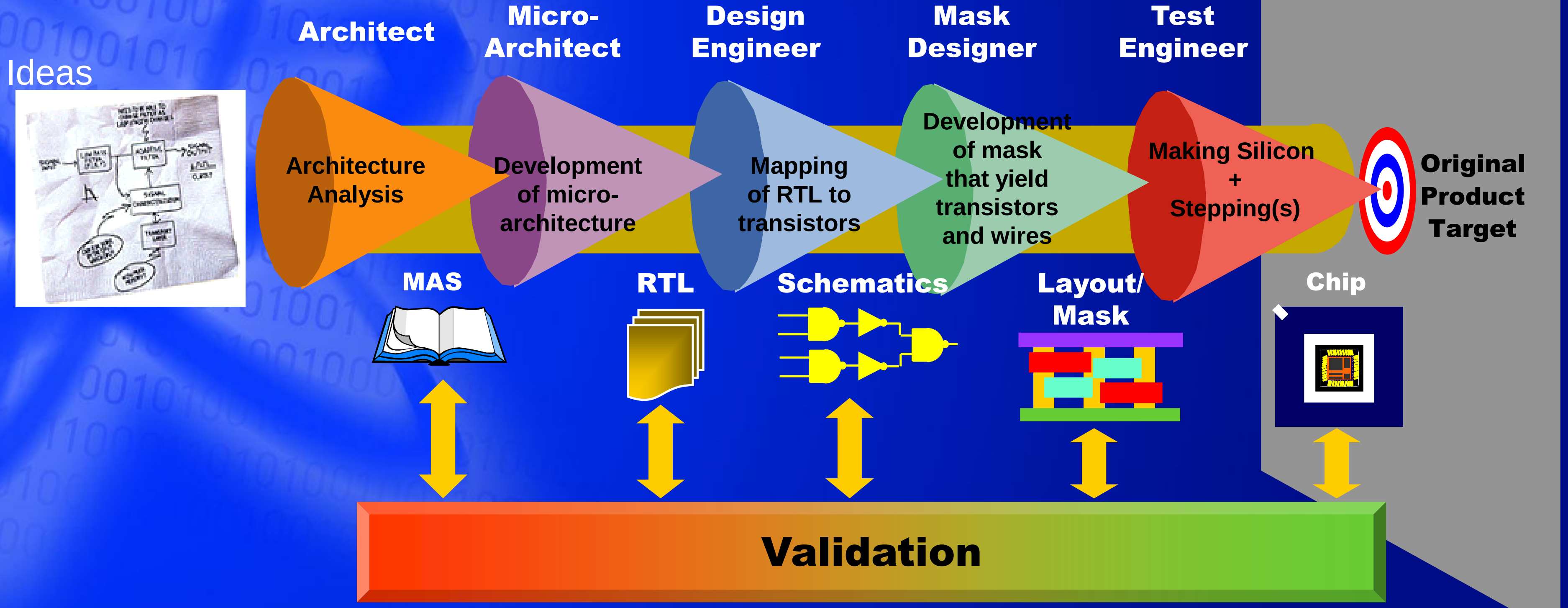
- Microsoft Azure (cloud services) combines 1 server CPU with 1 FPGA and all communication from the CPU to the network goes via the FPGA.
- Many algorithms have been sped up by factors between 10 and1000 times!

Today:

- Microsoft Azure (cloud services) combines 1 server CPU with 1 FPGA and all communication from the CPU to the network goes via the FPGA.
- Many algorithms have been sped up by factors between 10 and1000 times!
- However:

  At the same time, this design introduces new risks, since a bug or fault impacts the whole system. That, said [Microsoft Distinguished Engineer] Burger, has been the key challenge. "You are putting an alien technology into a very mature system. All of the network traffic runs through this thing. You screw it up, you can do some real damage.

Recall:

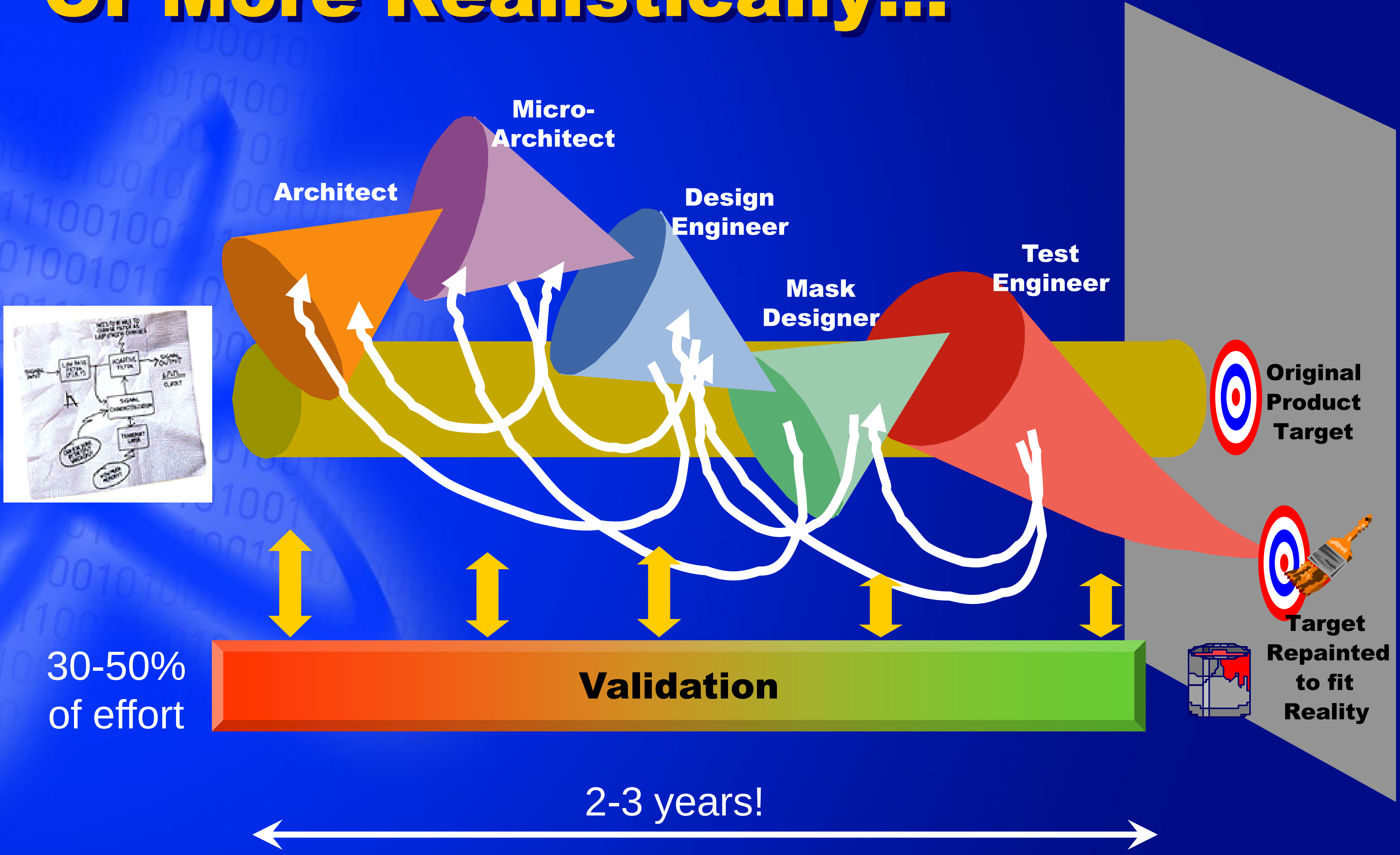


MAS: Micro-Architecture Specification

RTL: Register-Transfer Language

This is the theory…

Or More Realistically…

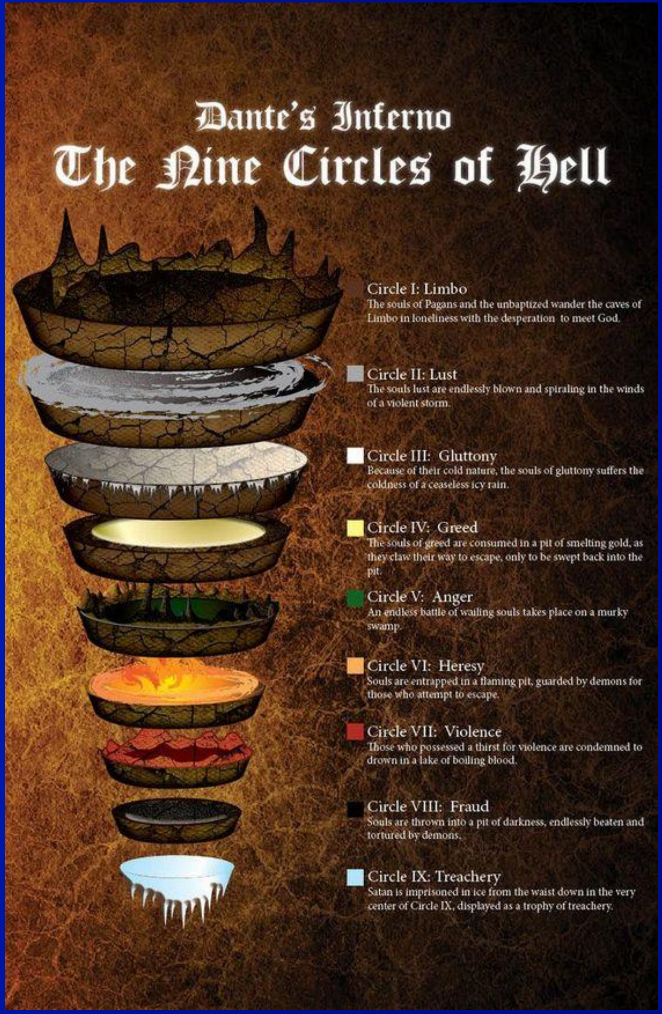


Challenge

- Create a good algorithm
- Partition it into SW and HW parts
- Implement SW part
  - Remember the critical communication link with the HW accelerator!
- Implement HW version
  - Re-design several times to achieve needed performance & size
- Debug HW
- Debug SW/HW system
- Profile resulting system
- Improve HW, improve SW, re-think partition, re-think algorithm
- Repeat…Repeat…Repeat…

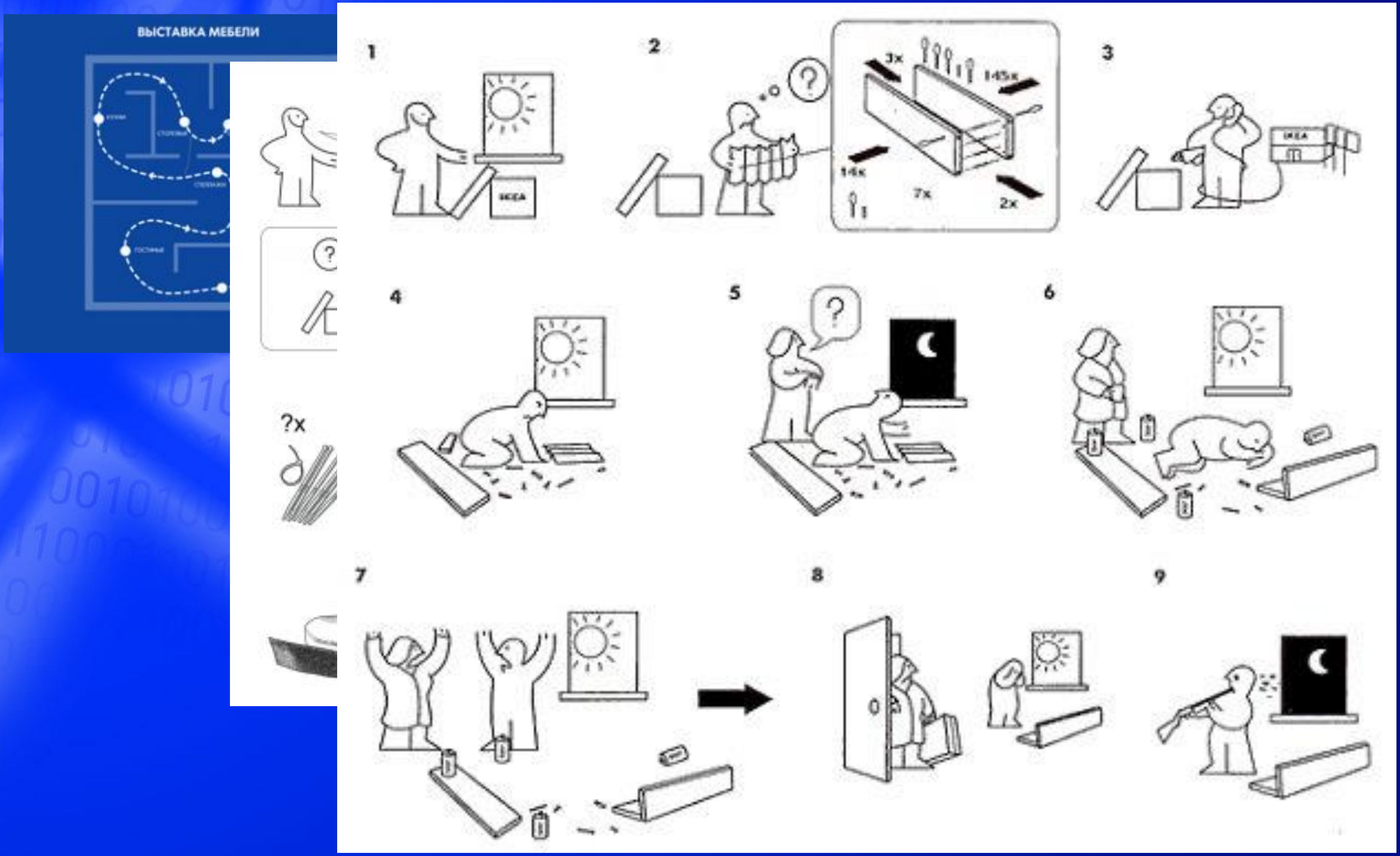
Bad News

- To verify HW designs is:
  - Hard
  - Time consuming
- To debug a HW design:
  - Is even worse!
- To debug combined SW/HW:
  - Is cause of short life span…
  - ..and lots of grey hair!


Dante's Inferno
The Nine Circles of Hell
Circle I: Limbo
The souls of Pagans and the unbaptized wander the caves of Limbo in loneliness with the desperation to meet God.
Circle II: Lust
The souls lust are endlessly blown and spiraling in the winds of a violent storm.
Circle III: Gluttony
Because of their cold nature, the souls of gluttony suffers the coldness of a ceaseless icy rain.
Circle IV: Greed
The souls of greed are consumed in a pit of smelting gold, as they claw their way to escape, only to be swept back into the pit.
Circle V: Anger
An endless battle of wailing souls takes place on a murky swamp.
Circle VI: Heresy
Souls are entrapped in a flaming pit, guarded by demons for those who attempt to escape.
Circle VII: Violence
Those who possessed a thirst for violence are condemned to drown in a lake of boiling blood.
Circle VIII: Fraud
Souls are thrown into a pit of darkness, endlessly beaten and tortured by demons.
Circle IX: Treachery
Satan is imprisoned in ice from the waist down in the very center of Circle IX, displayed as a trophy of treachery.

Good News!

- It could be worse…

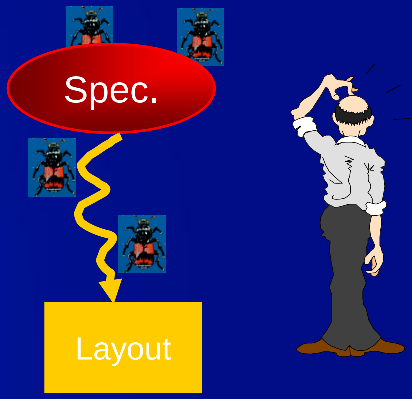


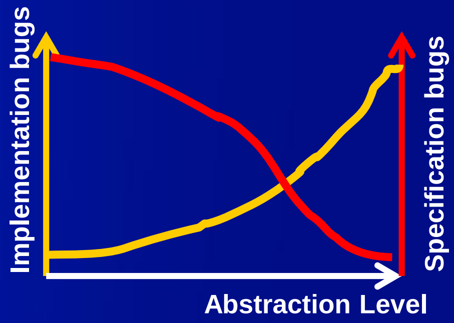
What Can be Done?

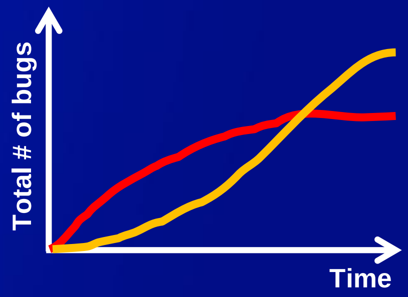
- Separate “what” from “how”
  - In practice, capture the algorithm at a high level of abstraction
- Use property driven verification/testing to ensure high-level model is “correct”.
- Rely on “correct-by-construction” for common tasks
  - Introducing the interface code between SW and HW is (almost) always the same. Automate its generation!
- Incorporate verification as part of the design process
  - No “design first, verify later” (if at all)!

Questions to be Answered

- How to capture desired functionality?
  - Language / level of abstraction
- How to ensure correct capture?
  - Property verification / validation
- How to refine the spec. to an imp?
  - Transformations / manual re-write
- How to ensure valid refinement?
  - FEV / correct by design

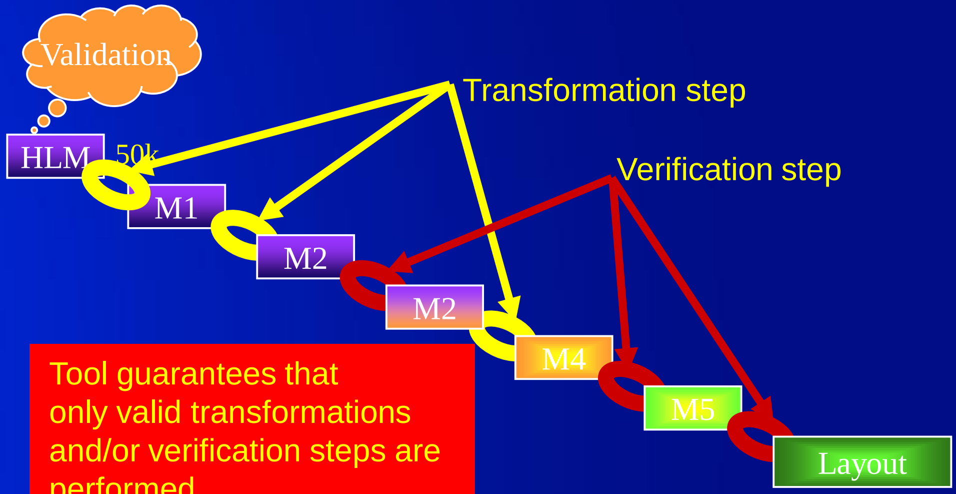




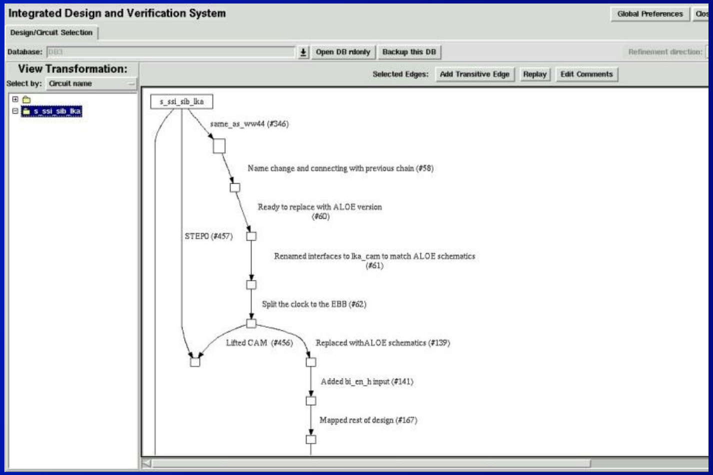


Integrate Design and Verification

- All validation work is reactive; the design gets created somehow and now we need to figure out if it is correct
- Rather than trying to do post-design verification, verify each step along the way.
  - Can mix "correct-by-construction" and "trust-but-verify" parts.
  - Can use different verification engines at different levels of abstraction
  - Imposes a relatively modest overhead on the design process for a big payoff.
  - A system can be built to track the "quality" of a design from correctness point of view.



Tool guarantees that only valid transformations and/or verification steps are performed

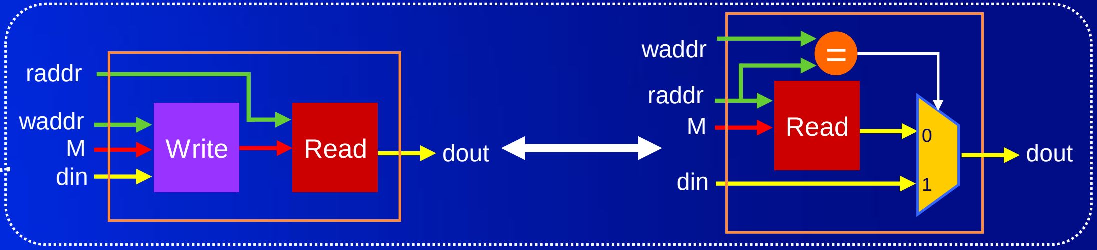


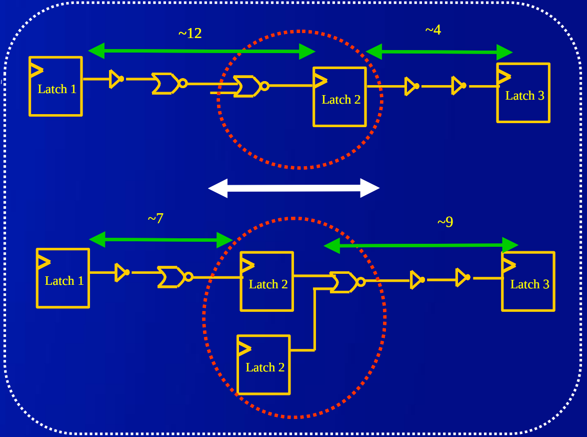
s_ssi_sib_lka
same_as_ww44 (#346)
Name change and connecting with previous chain (#58)
Ready to replace with ALOE version (#60)
STEP0 (#457)
Renamed interfaces to lka_cam to match ALOE schematics (#61)
Split the clock to the EBB (#62)
Lifted CAM (#456)
Replaced withALOE schematics (#139)
Added bi_en_h input (#141)
Mapped rest of design (#167)

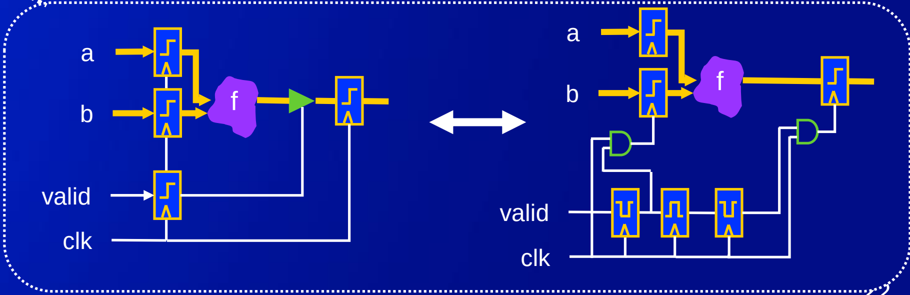
IDV prototype system for abstract RTL to layout with complete verification

Logical Design Transformations

- Add correct-by-construction implementation details
  - Examples:
    - Bypass
    - Re-timing
    - Duplication/merging of logic
    - Changing state encoding
    - Don't care usage
    - Introducing clock gating
    - …
- Allow arbitrary design changes when coupled with machine-checked justification

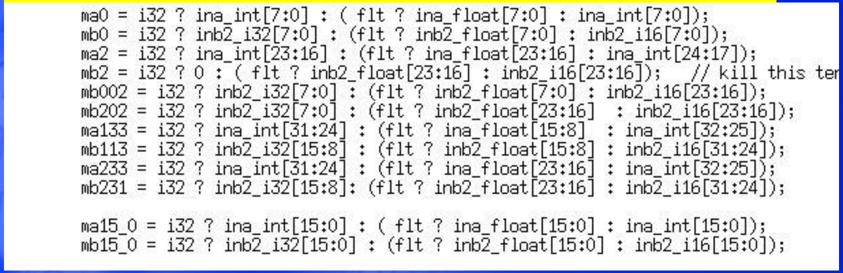






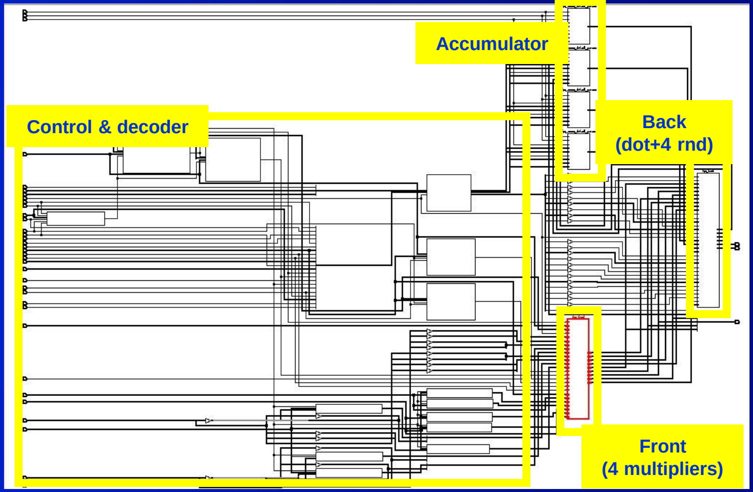
Example 1 From First IDV System

Graphics execution unit
High Level Model to Layout
HLM/aRTL lines of code + 20 pages tables

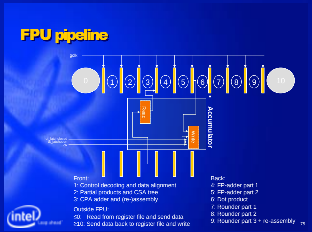


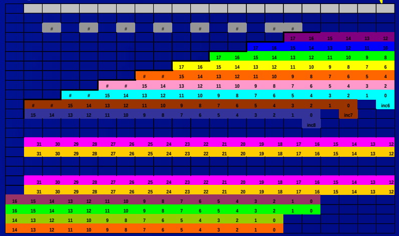
```
ma0 = i32 ? ina_int[7:0] : ( flt ? ina_float[7:0] : ina_int[7:0]);
mb0 = i32 ? inb2_i32[7:0] : (flt ? inb2_float[7:0] : inb2_i16[7:0]);
ma2 = i32 ? ina_int[23:16] : (flt ? ina_float[23:16] : ina_int[24:17]);
mb2 = i32 ? 0 : ( flt ? inb2_float[23:16] : inb2_i16[23:16]);   // kill this ter
mb002 = i32 ? inb2_i32[7:0] : (flt ? inb2_float[7:0] : inb2_i16[23:16]);
mb202 = i32 ? inb2_i32[7:0] : (flt ? inb2_float[23:16]  : inb2_i16[23:16]);
ma133 = i32 ? ina_int[31:24] : (flt ? ina_float[15:8]  : ina_int[32:25]);
mb113 = i32 ? inb2_i32[15:8] : (flt ? inb2_float[15:8] : inb2_i16[31:24]);
ma233 = i32 ? ina_int[31:24] : (flt ? ina_float[23:16] : ina_int[32:25]);
mb231 = i32 ? inb2_i32[15:8]: (flt ? inb2_float[23:16] : inb2_i16[31:24]);

ma15_0 = i32 ? ina_int[15:0] : ( flt ? ina_float[15:0] : ina_int[15:0]);
mb15_0 = i32 ? inb2_i32[15:0] : (flt ? inb2_float[15:0] : inb2_i16[15:0]);
```

High-level specification

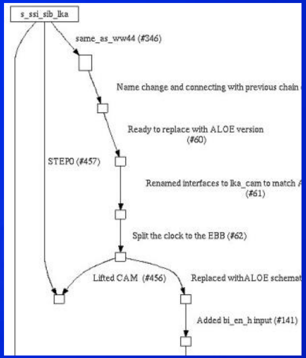


New implementation algorithm ideas

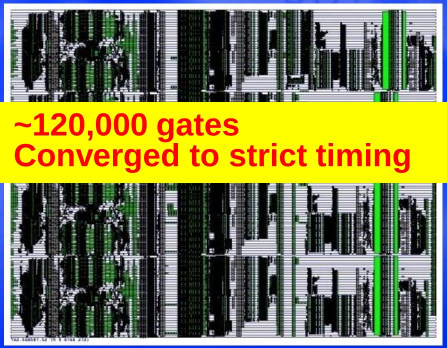

FPU pipeline
Accumulator
Front:
1: Control decoding and data alignment
2: Partial products and CSA tree
3: CPA adder and (re-)assembly
Outside FPU:
≤0: Read from register file and send data
≥10: Send data back to register file and write
Back:
4: FP-adder part 1
5: FP-adder part 2
6: Dot product
7: Rounder part 1
8: Rounder part 2
9: Rounder part 3 + re-assembly
intel



Design and verification in IDV


s_ssi_sib_lka
same_as_ww44 (#346)
Name change and connecting with previous chain
Ready to replace with ALOE version (#60)
STEP0 (#457)
Renamed interfaces to lka_cam to match (#61)
Split the clock to the EBB (#62)
Lifted CAM (#456)
Replaced with ALOE schemat
Added bi_en_h input (#141)

Final placed result

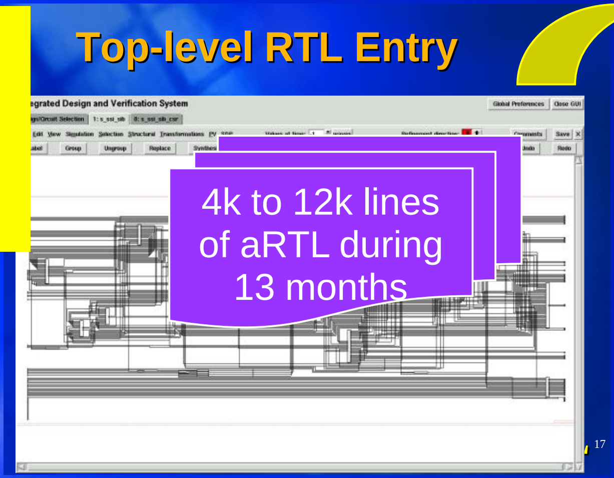


~120,000 gates
Converged to strict timing

Bottom line: It actually works!

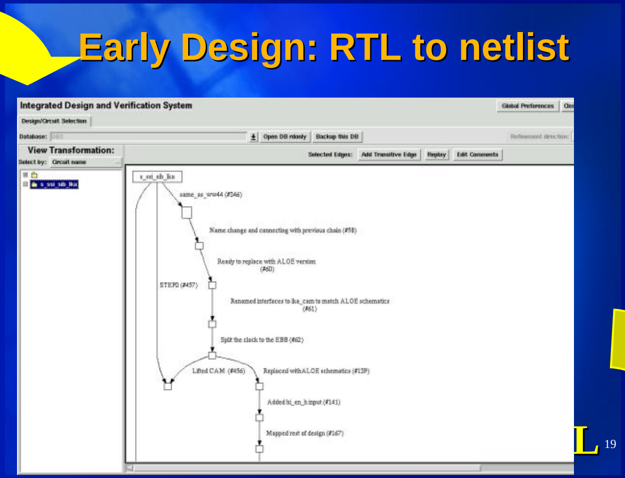
Example 2 From First IDV System

3 designers (instead of 8)
25 FUBs
5 RF, 3 CAM EBBs
In production flow for more than 1 year



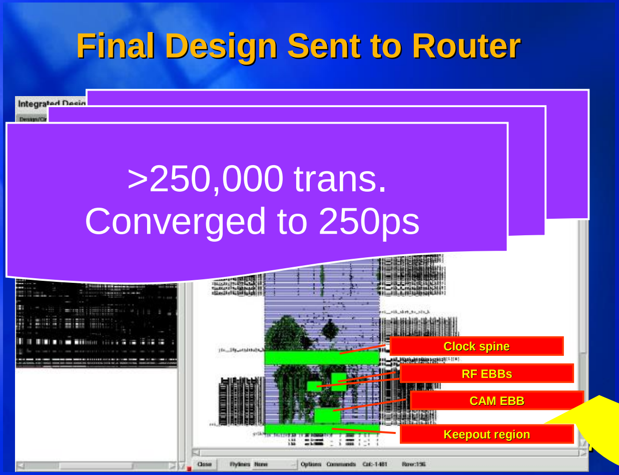
Top-level RTL Entry

4k to 12k lines of aRTL during 13 months



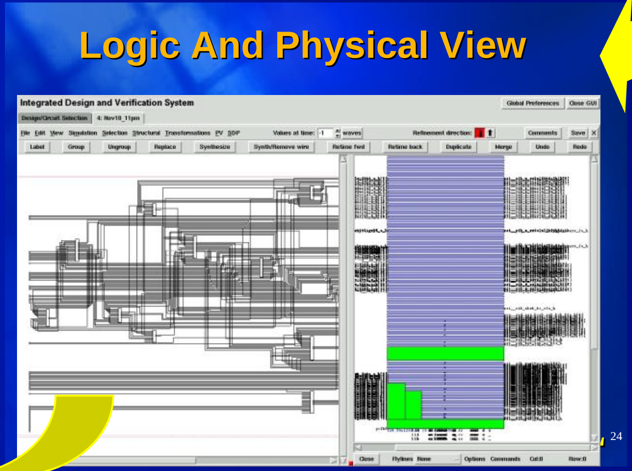
Early Design: RTL to netlist

Integrated Design and Verification System
Global Preferences
Open DB rdonly
Backup this DB
View Transformation:
Selected Edges:
Add Transitive Edge
Replay
Edit Comments
s_ssi_sb_lka
same_as_ww44 (#346)
Name change and connecting with previous chain (#58)
Ready to replace with ALOE version (#60)
STEP0 (#457)
Renamed interfaces to lka_cam to match ALOE schematics (#61)
Split the clock to the EBB (#62)
Lifted CAM (#456)
Replaced with ALOE schematics (#139)
Added bi_en_h input (#141)
Mapped rest of design (#167)
19



Final Design Sent to Router

>250,000 trans.
Converged to 250ps



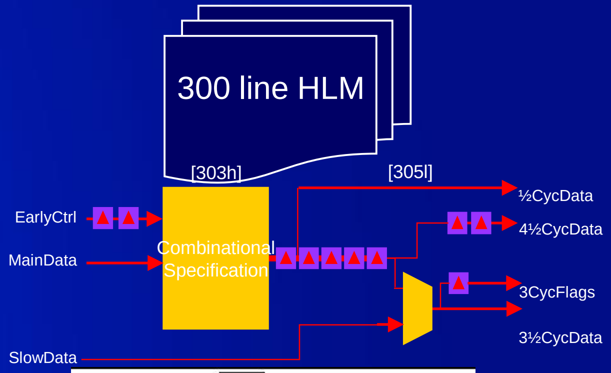
Logic And Physical View

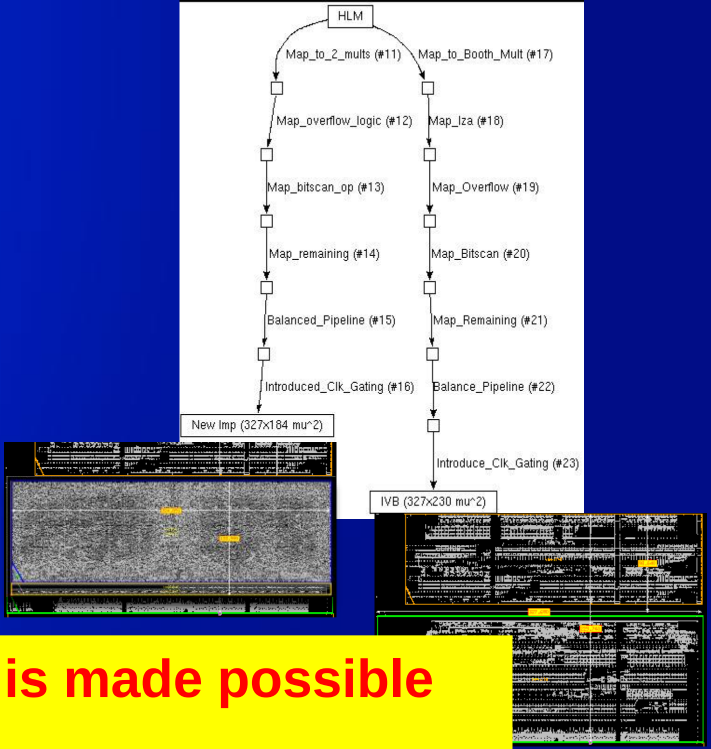
Bottom line: During 13 months of design effort, no aRTL changes were needed because of implementation considerations.

Example 3 From First System

- Integer multiplication unit
  - ➢ RTL (“How”)
    - - >3,000 lines
  - ➢ HLM (“What”)
    - - <300 lines

- Two implementations derived inside IDV
  1. To the existing RTL implementation
  2. New version using a different algorithm and partitioning
  - ➢ New version was 20% smaller than original version
  - ➢ Both provably equal to HLM and thus HLM validation was shared.





Bottom line: Rapid design exploration is made possible without extra verification cost.

Lessons Learned

- Integrating Design and Verification:
  - Is technically entirely feasible
    - Requires fairly significant system to be built for approach to be practical.
    - Rapidly changing specifications are challenging, but doable.
  - Allowed far more design exploration
    - First implementation took "normal" time
    - Second, third, … versions took only a fraction of initial design time.
  - Requires a completely different mentality
    - Combines two roles (design engineer and verification engineer)
    - Requires a new approach to teaching design & verification
- IDV idea failed to be widely deployed inside Intel
  - Project eventually cancelled.
  - Likely ahead of its time…

Why Do it Again?

"Insanity is doing the same thing, over and over again, but expecting different results."

- Narcotics Anonymous

- The short design cycle ideal for IDV
  - Trying multiple alternatives not only useful, but necessary
- The user community is entirely different
  - Training in HW design is required from day one
  - No legacy "style" in place to tear down.
- FPGA based design require much less physical design work
  - A major part of the original IDV system devoted to physical design
  - 2/3 of transformations were related to physical design aspects
- Great need for efficient techniques for developing these types of accelerated applications!

Some Further Research Questions

- What transformations do we need in the SW domain?
- What decision procedures are needed for SW refinements?
- How does an efficient "split into SW+HW" transformation look like?
  - Must it be "trusted" or can it be verified (added flexibility)
- How do we train "vertical developers" that can move seamlessly between SW and HW?

Conclusion

Integrate Design & Verification:

==

Catch the bugs as soon as they are created!

Thank you!

Questions?