

***Managing the Transition from  
Complexity to Elegance***

**Charles Moore**

Senior Research Fellow  
Department of Computer Sciences  
The University of Texas at Austin  
[crmoore@cs.utexas.edu](mailto:crmoore@cs.utexas.edu)

# ***Top 10 Indicators that your Project Might Have Complexity Issues***

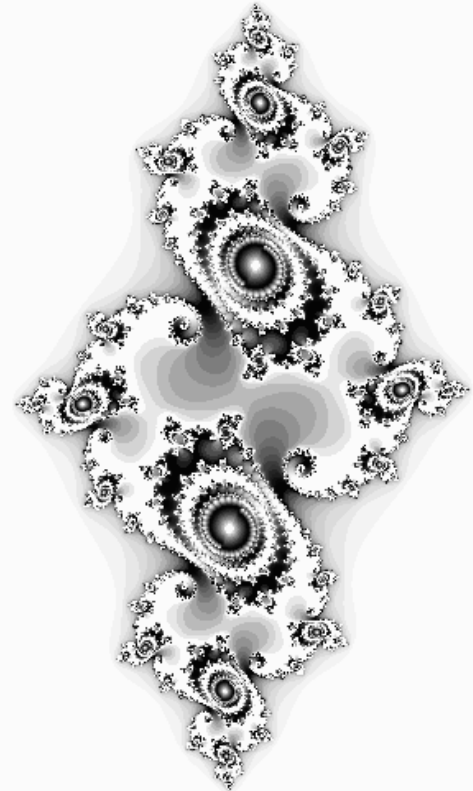
---

1. Several individuals on your team have filed in excess of 100 patents!
2. Designer says, *“It really is simple ... I just can’t explain it to you...”*
3. The number of operational modes approaches the number of instructions
4. Designer says, *“What knee of the curve?”*
5. Design team has 5 different phrases for talking about the same thing
6. Designer says, *“Let’s get the function right first, then worry about those other things”*
7. Most “design fixes” result in one or more *new* bugs
8. You find large triple-nested `case` statements in the HDL
9. Your architects outnumber your verification people
10. You have a daily meeting to discuss new requirements

# Overview

---

- What is Complexity-effective Design?
- Power4 Story
  - Lessons from the Alpha design team
  - Key design principles
  - Design Process
- Looking toward the Future
  - Evolutionary Approach
  - Revolutionary Approach
- TRIPS: New design principles
- Concluding Comments



# Definition of Complexity Effective Design

---

- ***Workshop Organizer's Definition:***

A complexity-effective design feature either:

(a) yields a significant performance (and/or power efficiency) improvement relative to the increase in hardware/software complexity incurred; or

(b) significantly reduces complexity (and/or power consumption) with a tolerable performance impact.

- ***My Augmentation:***

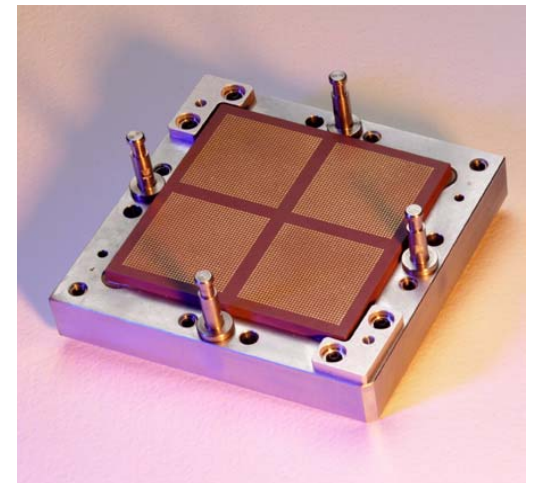
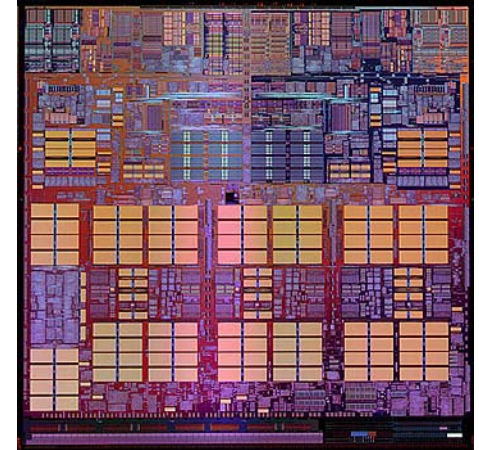
A complexity-effective design:

(1) Embraces a relatively small set of overriding design principles and associated mechanisms

(2) Has been ruthless in collapsing unnecessary complexity into these more fundamental and elegant mechanisms

# POWER4: First Requirements

- Full system design
  - Start with *system-level* requirements and constraints
  - 32-way SMP – *not just a microprocessor*
  - Scalable balance - *adding CPUs also adds cache & mem*
  - RAS, Error Handling, System Management
- Innovative, aggressive storage system design
  - Heavy investment in system level bandwidth
  - Multi-level shared caches
  - Hundreds of concurrent system-level transactions
- Optimized OS, Middleware and Applications
  - Binary compatibility
- Methodology & process for continued leadership
  - High frequency design
  - Design verification



# Lessons from the Alpha Design Team

---

**EV-6:** Excellent example of complexity-effective superscalar

- Establish a solid baseline, and learn to say **NO!**
  - Optimize for the common cases
  - Force less common cases to make use of existing mechanisms
  - >5% gain *and* less than a month to implement? If not, *forget about it.*
- Keeping it simple enables *high frequency custom design*
  - Architecture, Micro-architecture and Methodology
  - Designers focus on custom macros instead of wicked logic puzzles
- Find and leverage inherent technology / circuit / function alignment
  - SRAM-like regularity – *CAMs, Dot-OR mux, etc.*
  - Make control logic as “dataflow-like” as possible
- Limit the number of logic designers (architects) to less than 10
  - Invest in logic savvy circuit designers, and verification people

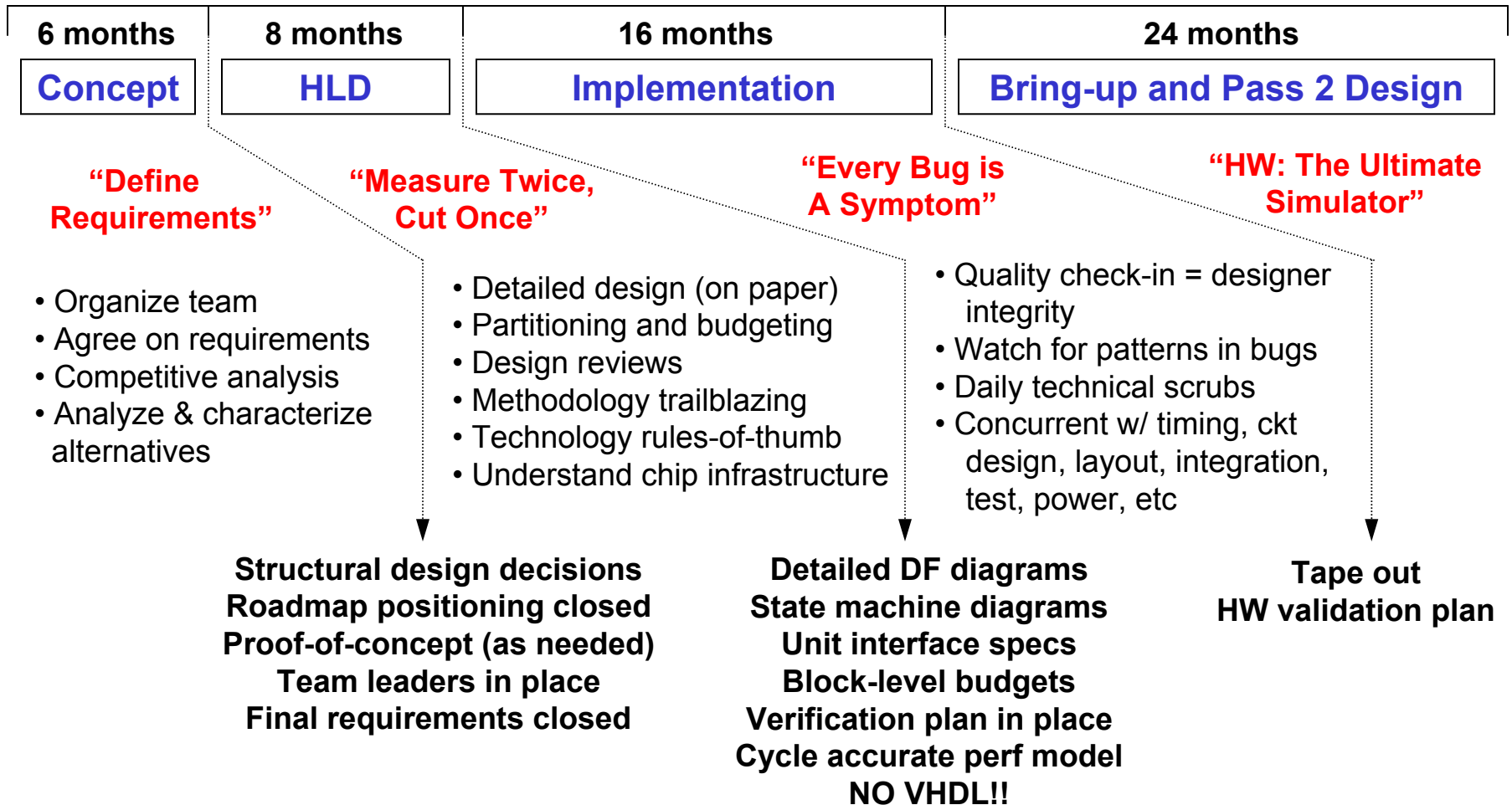
# POWER4: Key Design Principles in the Core

---

Strongly Influenced by Lessons Learned from Alpha

- Full Custom Design
  - Apply to CPU, L2 and L3 – *balance and scalability at system-level*
- “Knee of the curve” out-of-order superscalar
  - 2P CMP versus 1 wider-issue CPU – *balance ILP and TLP*
  - Out-of-order – *justified primarily for latency tolerance*
- “Layering” – *keep inner core as simple as possible*
  - Instruction cracking – *inner core only sees streamlined instructions*
  - Non-stop pipeline - *minimize random control logic*
  - Invest in *issue-queue-retry* and *pipeline-flush* mechanisms
  - Coherency – *push cache coherency burden into L2 controllers*
- Commercial *and* Technical workloads
  - Two double precision FPUs – *provide computation power as needed*
  - L1 Dcache 2R/1W per cycle – *sustain LD LD FMA ST BC every cycle*
  - Hardware stride detection and prefetch controller – *data!*

# POWER4: Phased Design Process for Managing Complexity



**Huge Investment Up Front – Naturally Filters Out Complexity**

# Looking Toward the Future (1)

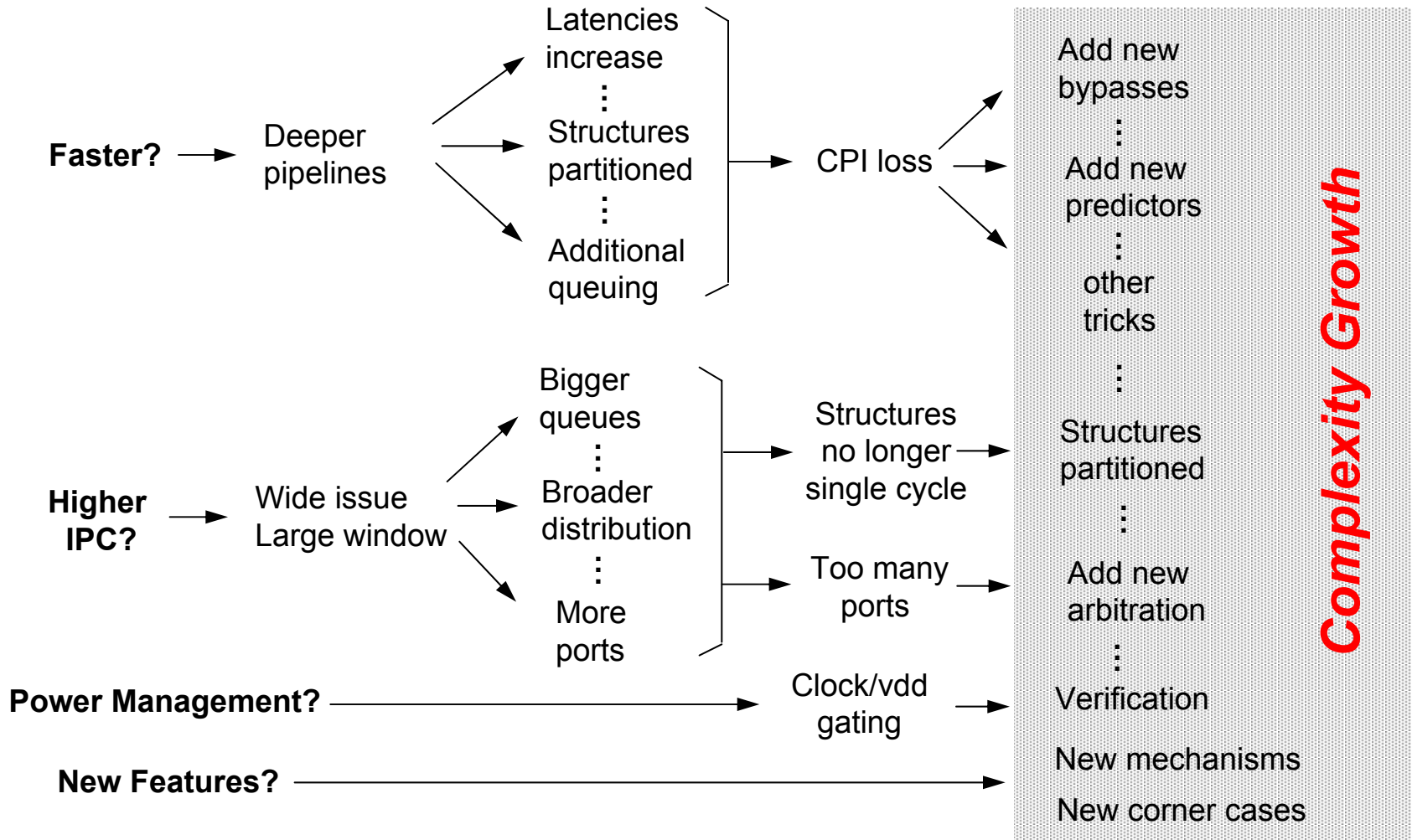
---

## 1. Evolutionary Approach - *Build on what you already have*

- Stick with originally conceived fundamentals
  - Work new features/requirements in while minimizing disruption
- Evolution of superscalar (1992-2003):
  - Enormous gains in frequency: 200MHz → 3GHz
  - Minimal CPI gains: remains at ~1.0, despite considerable effort
  - Complexity is growing quickly – Verification is the gate

**Key Question:** *What are the implications of multi-generational evolution?*

# Evolution and the “Complexity Spiral”



**Superposition Applies ... complexity tends to compound!**

# Looking Toward the Future (2)

---

## 2. Revolutionary Approach - *Start with a clean sheet of paper*

- Identify new fundamentals to carry design for next 10 years
  - Address emerging technology and business trends
  - Embrace new abstractions
- Many barriers:
  - Need significant benefit over competing evolutionary approach
  - Compatibility
  - Market timing

**Key Question:** *What are the key technical and business trends that might justify a revolutionary approach?*

# Emerging Technical and Market Trends

---

## Emerging Sources of Complexity:

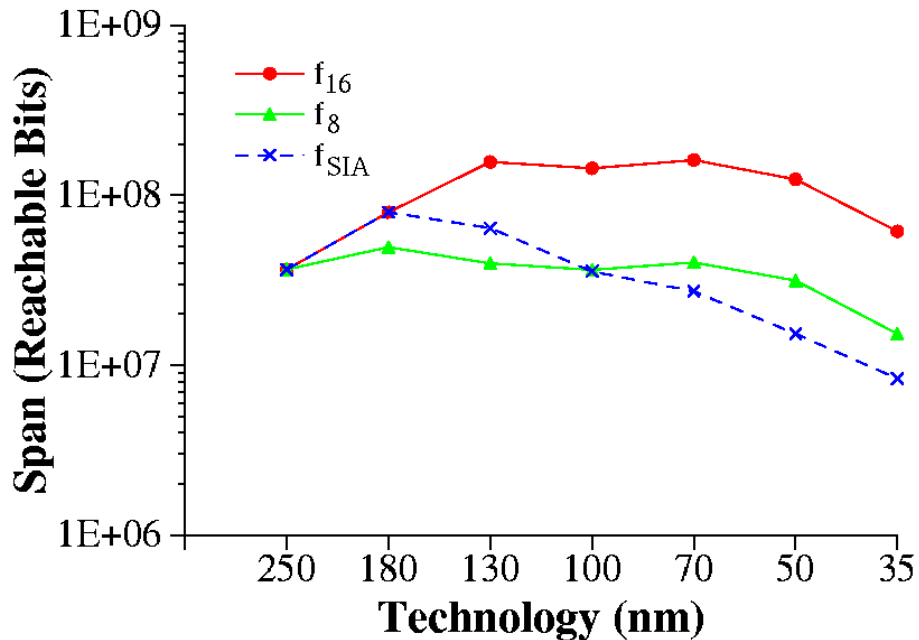
1. Wire delay
2. Cycle time
3. Power Trends – static and dynamic
4. Workload Diversity
5. Soft errors and Reliability

## Emerging Implications of Complexity:

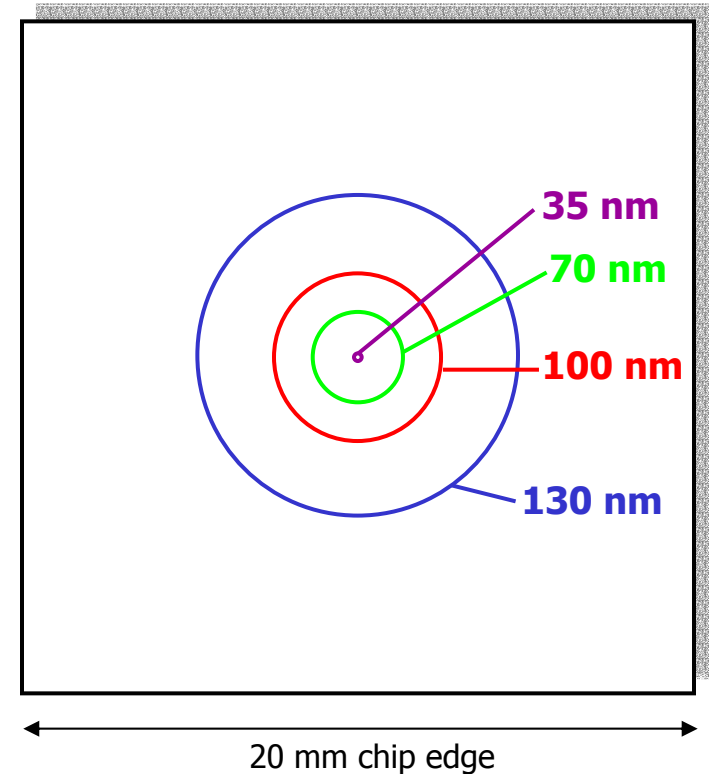
6. Overhead circuitry (vs. ALU circuitry)
7. Designer Productivity and Cost
8. Mask Cost

# 1: Wire Delay in Future Technology

Analytically ...



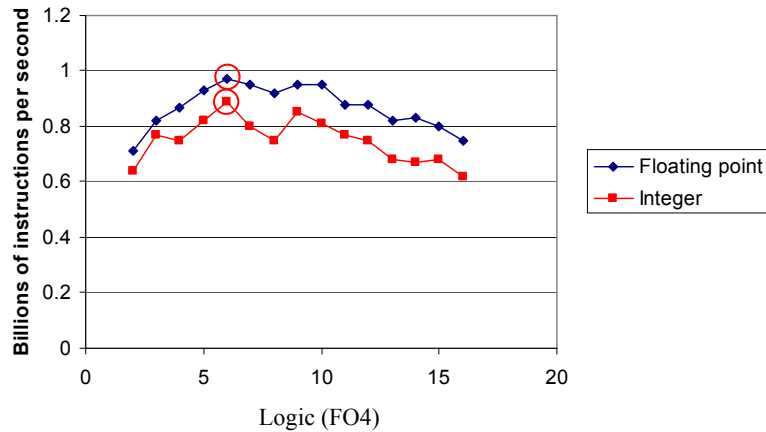
Qualitatively ...



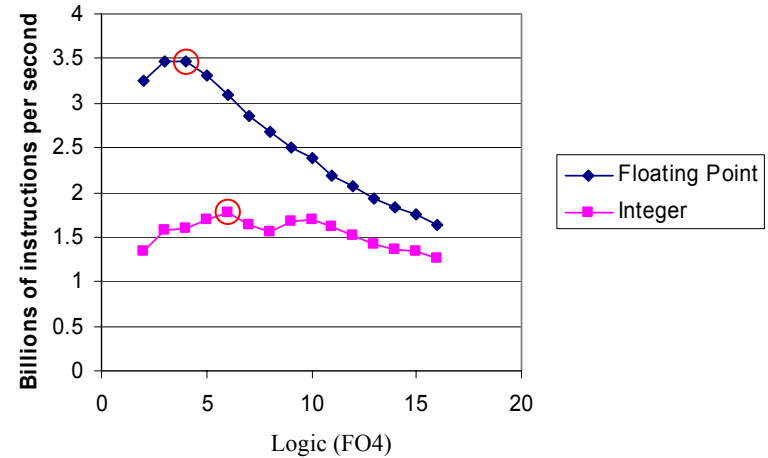
**Either way ... Partitioning for on-chip communication is key**

# 2: Limits on Pipelining and Frequency

### In-Order Processor



### Dynamically Scheduled Processor



$$\text{Total (FO4)} = \text{Logic (FO4)} + \text{Overhead (1.8 FO4)}$$

Machine Type	Integer Code	FP Code
In-Order Processor	8 FO4	8 FO4
Out-of-Order Processor	8 FO4	6 FO4
Cray-1S <sup>1</sup>	10.9 FO4	5.4 FO4

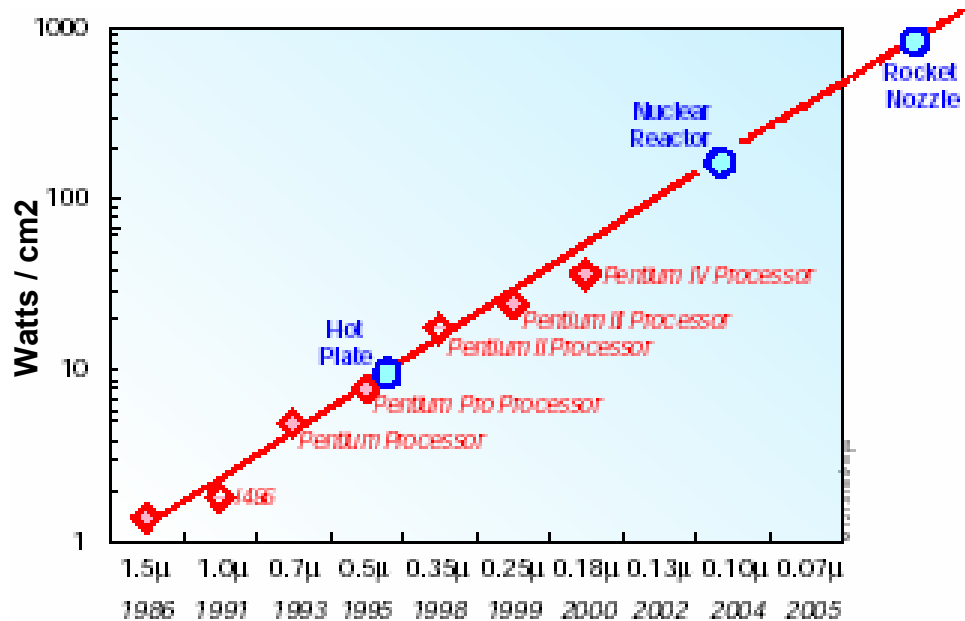
**Current designs at 18-20 FO4  
Only ~2X improvement  
remains.**

<sup>1</sup>Kunkel and Smith [ISCA '86]

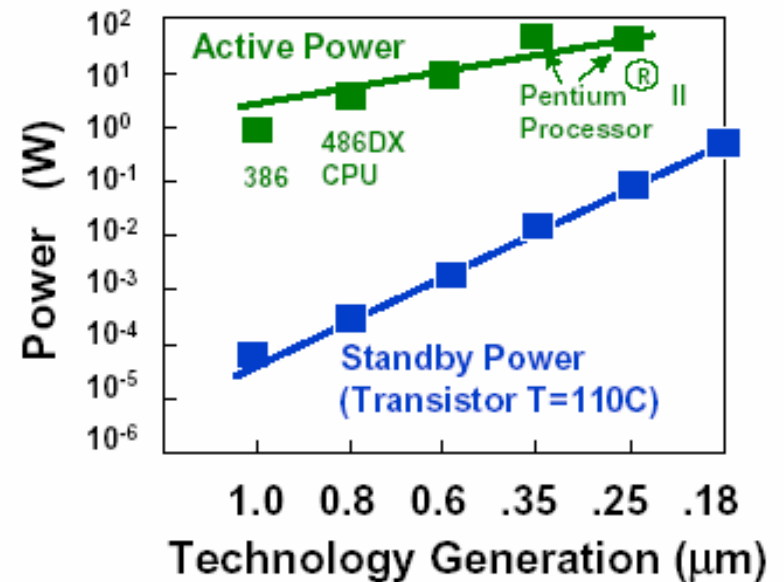
**After that, Frequency growth limited to raw Technology  
Q: What about Uni-processor performance?**

# 3: Power Trends

## Power Density



## Leakage Power



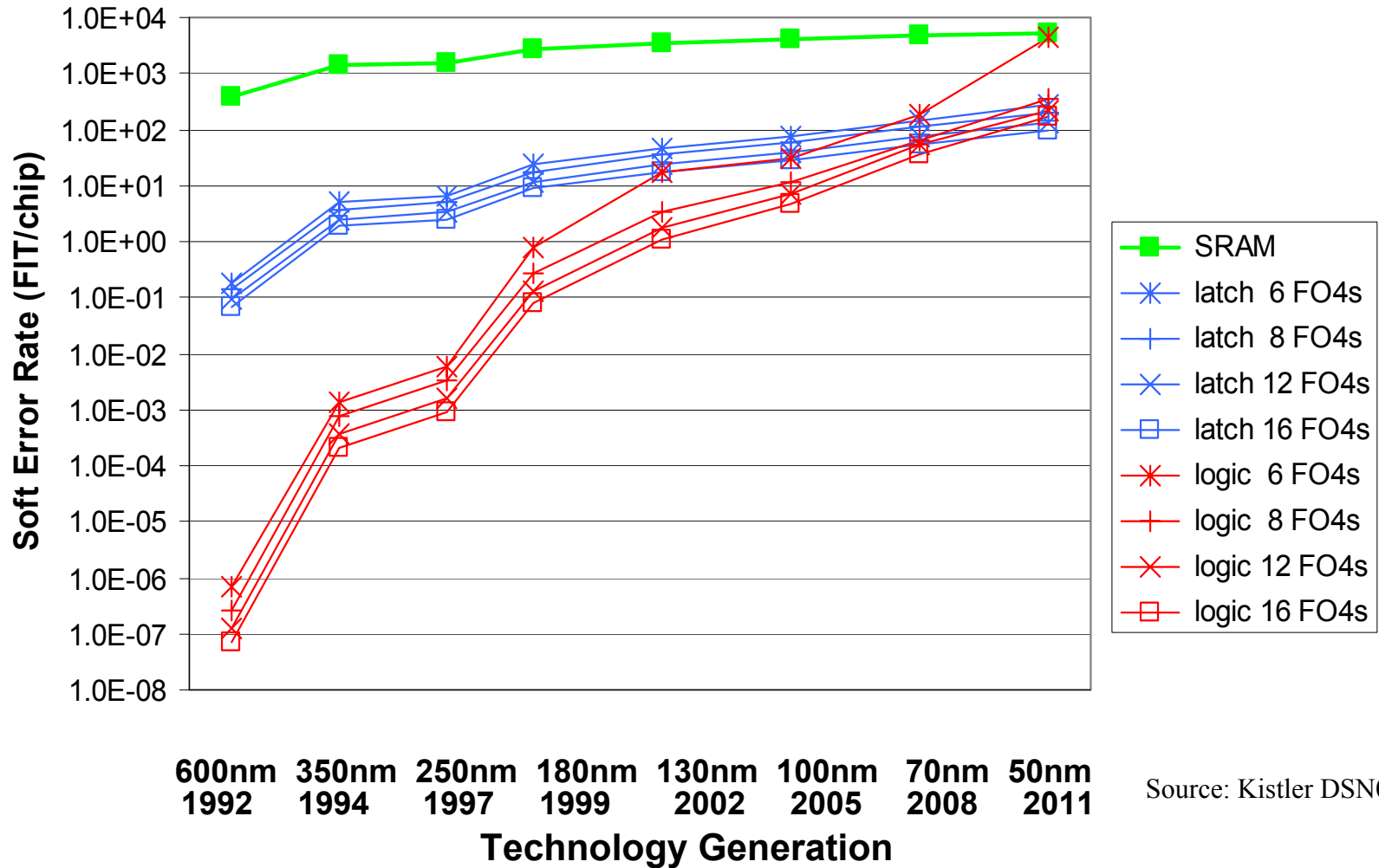
Power is now a first order design constraint

# 4: Workload Diversity Trends

---

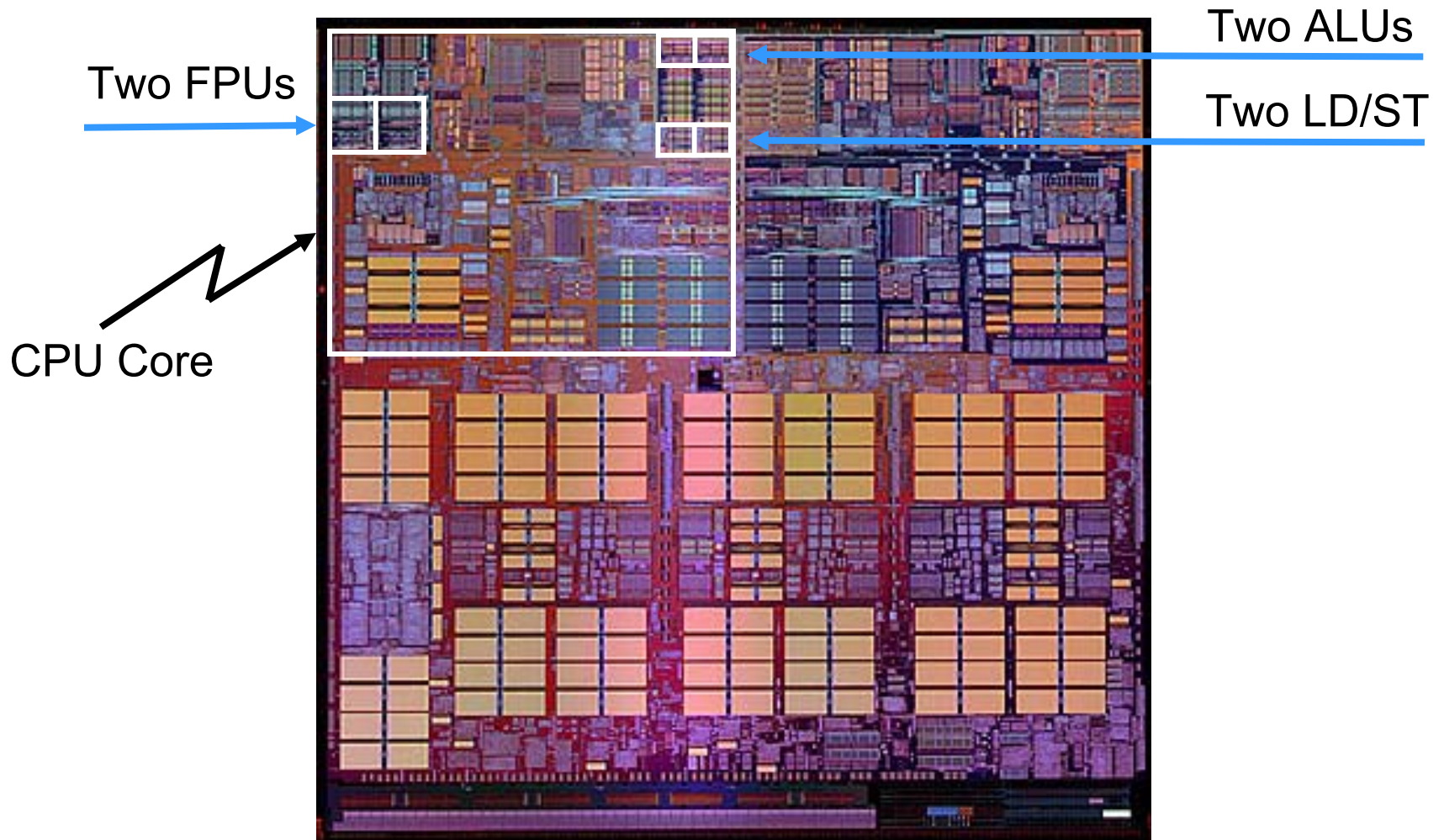
- Workloads are becoming more diverse:
  - Streaming applications need enormous bandwidth
  - Threaded workloads need throughput and efficient communication
  - Vector workloads need large execution resources
  - Desktop applications need powerful uni-processors
- Advanced applications show different behavior in different phases
  - Image recognition and tracking:
    - Signal processing (filtering, image processing)
    - Server (image recognition, database search)
    - Sequential (decision support, planning)
  - Streaming video servers
  - OS versus User code
- General purpose machines becoming more “fragile”
  - Many things must be aligned to get best performance
  - Anomalies are common

# 5: Chip Soft Error Rate (SER) Trends



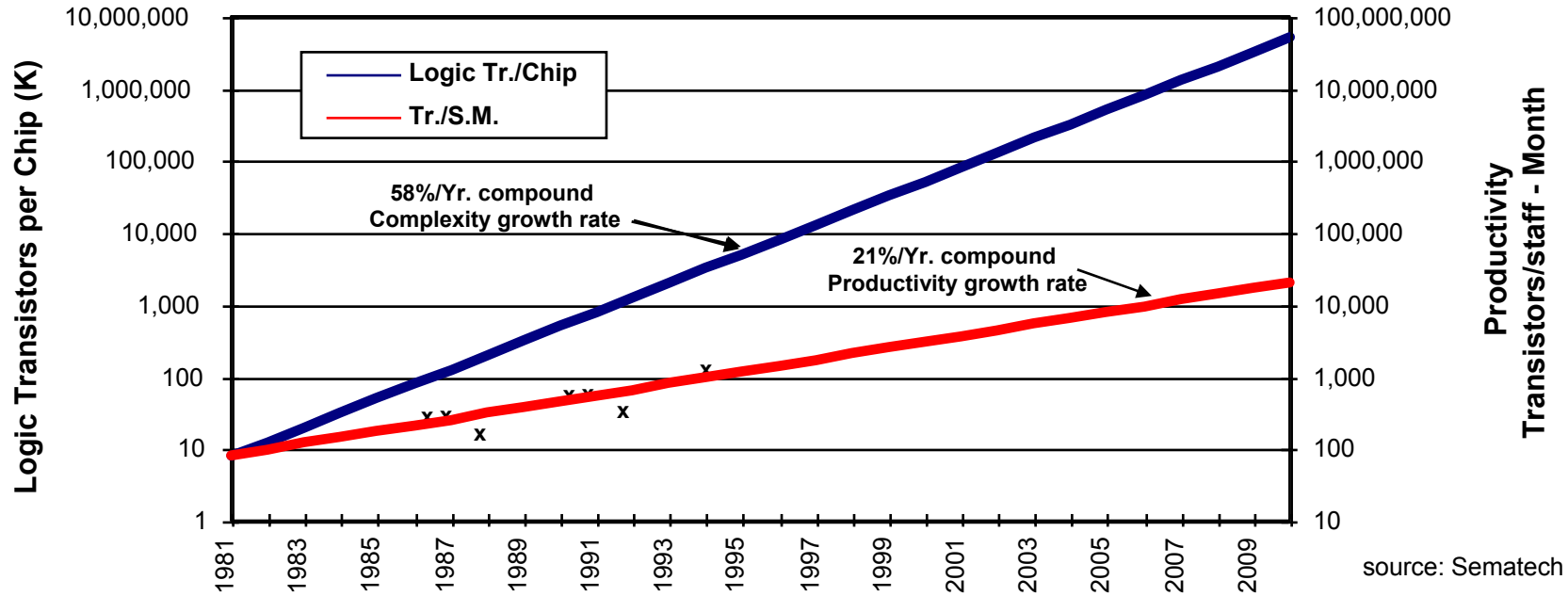
Source: Kistler DSN02

## 6: Overhead – “Spot the ALU”



**Only 12% of Non-Cache, Non-TLB Core Area is Execution Units**

# 7: Design Productivity Trends



source: Sematech

<u>Year</u>	<u>Technology</u>	<u>Chip Complexity</u>	<u>Total Staff Yrs.</u>	<u>3 yr Design Staff</u>	<u>Staff Cost</u>
2001	180 nm	80 M Tr.	1800	600	\$260 M
2003	130 nm	200 M Tr.	2600	875	\$360 M
2005	90 nm	680 M Tr.	5400	1800	\$750 M
2007	65 nm	1000 M Tr.	8300	2800	\$1.16 B

## 8: Mask Cost Trends

---

Technology	Mask Cost
250nm	\$100K *
180nm	\$350K *
130nm	\$750K *
90nm	\$1.5M *
65nm	\$3M **
35nm	\$6M **

Source:  
\* Sematech  
\*\* My estimate (2X/node)

**Double Jeopardy!** 1) The potential for bugs goes up  
2) The cost of re-spinning the chip goes up

**Implication:** *You Can't Afford Hardware Bugs!!*

# TRIPS: Taking the Revolutionary Path

---

## Emerging Sources of Complexity:

1. *Wire delay*
2. *Cycle time*
3. Power Trends – static and dynamic
4. *Workload Diversity*
5. *Soft errors and Reliability – (in progress)*

## Emerging Implications of Complexity:

6. *Overhead circuitry (vs. ALU circuitry)*
7. *Designer Productivity and Cost*
8. Mask Cost

# TRIPS: Overriding Design Principles

---

- **Minimize Centralized and Associative Structures**

  - [Overhead Circuitry] – eliminate structures dominating today’s designs

  - [Wire Delay] – minimize long global wires

- **Block-oriented Dataflow Execution Model**

  - [Cycle Time] – extend single-thread performance after frequency peaks

- **Polymorphism – Support for workload diversity**

  - [Workload Diversity] – perform well on ILP, TLP, DLP workloads

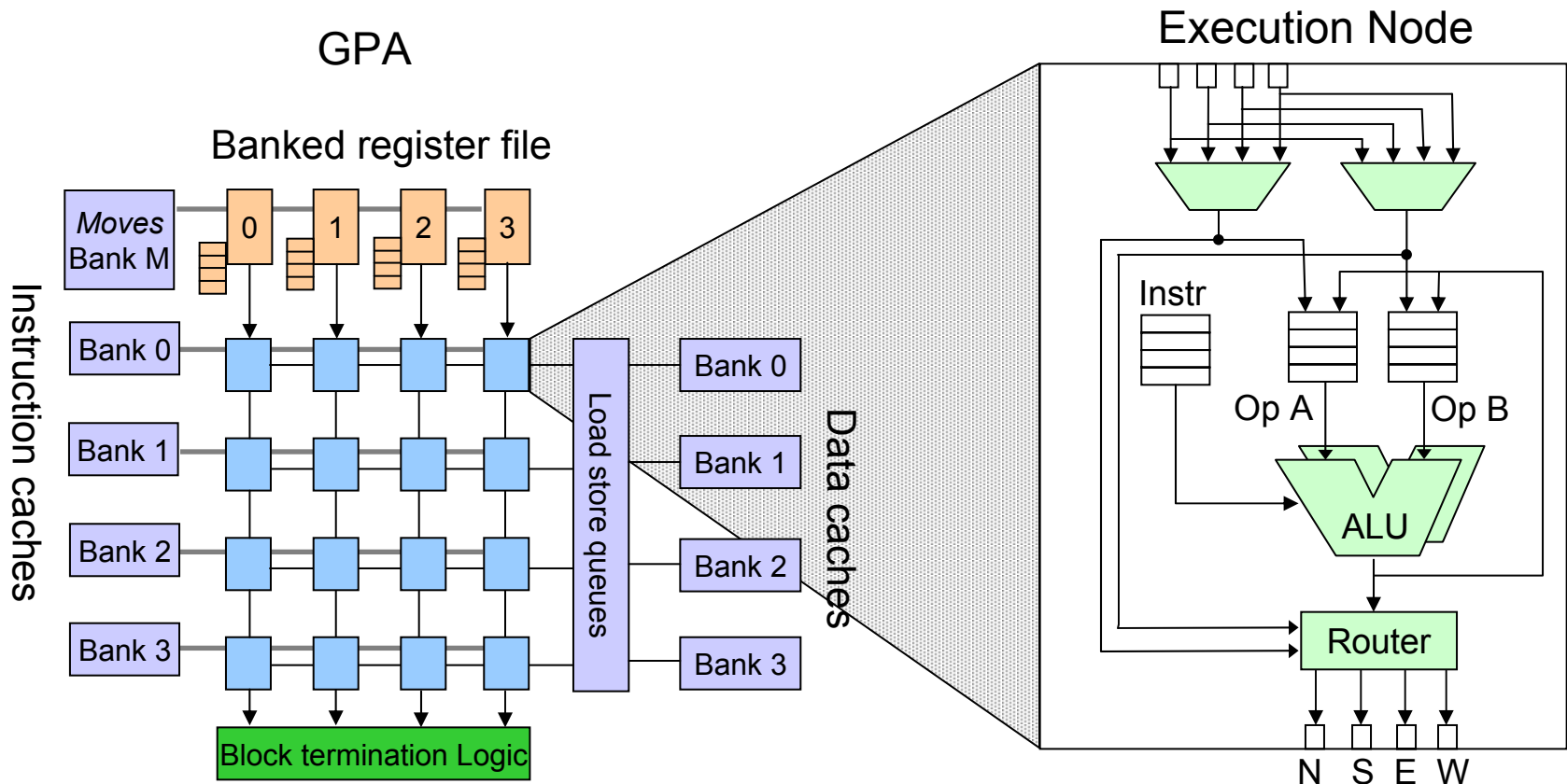
  - [Reliability] – “morphware” that tolerates *fail-in-place* components

- **“Architectural” Partitioning**

  - [Wire Delay] – pre-planned 1-cycle communication between neighbor tiles

  - [Designer Productivity] – Regularity and re-use of physical components

# TRIPS: Grid Processor Overview



- Overhead Circuitry Eliminated
- ▶ No associative issue queues
  - ▶ No global bypass
  - ▶ No hardware dependency analysis
  - ▶ No internal register renaming

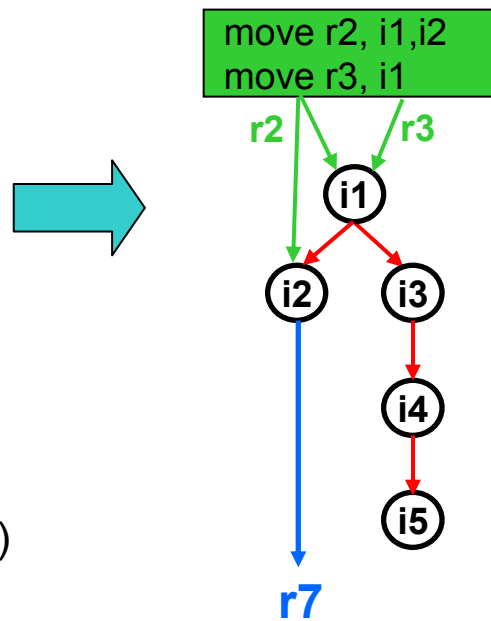
# Block Compilation

## Intermediate Code

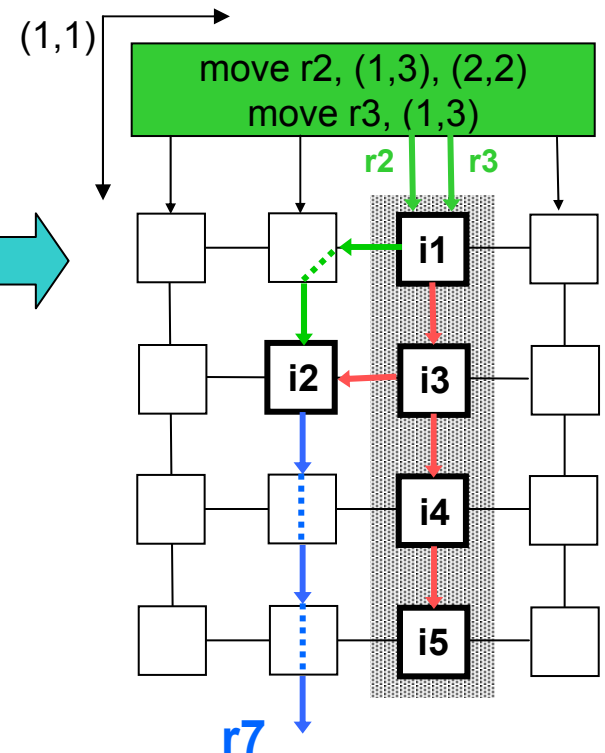
i1) add r1, r2, r3  
i2) add r7, r2, r1  
i3) ld r4, (r1)  
i4) add r5, r4, 1  
i5) beqz r5, 0xdeac

- Inputs (r2, r3)
- Temporaries (r1, r4, r5)
- Outputs (r7)

## Data flow graph



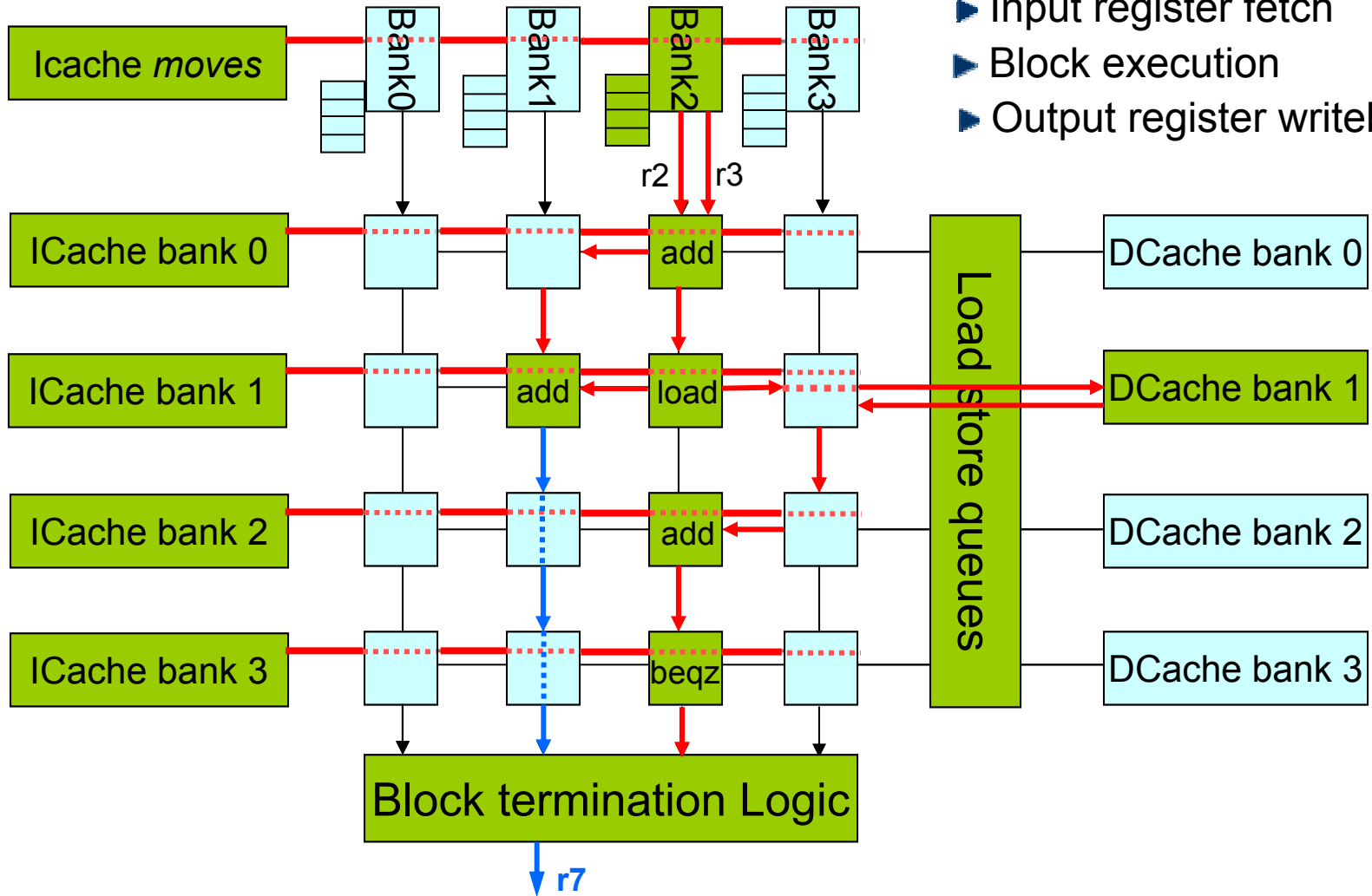
## Mapping onto GPA



First, place **critical path** to minimize communication delays  
Then place less critical paths to *maximize ILP*

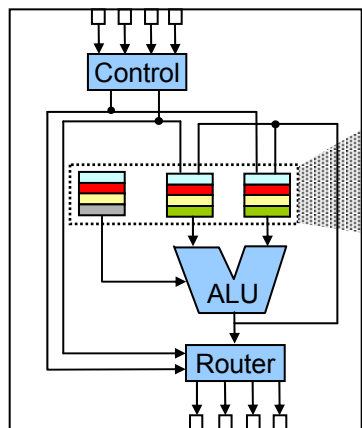
# Block Execution

- ▶ Instruction distribution
- ▶ Input register fetch
- ▶ Block execution
- ▶ Output register writeback

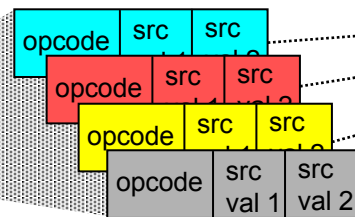


# Instruction Buffers - *frames*

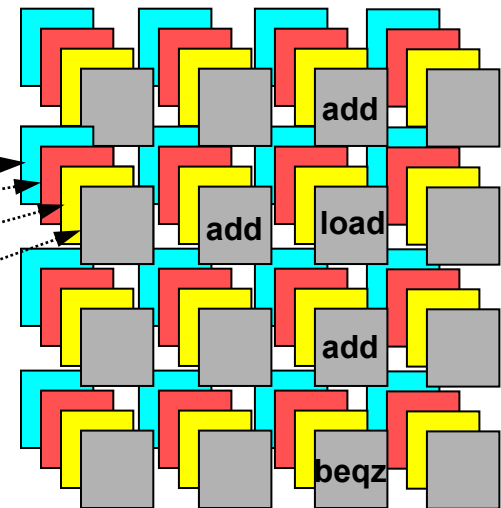
- Instruction Buffers add *depth* and define *frames*
  - 2D GRID of execution units; 3D scheduling of instructions
  - Allows very large blocks to be mapped onto GRID
  - Result addresses explicitly specified in 3-dimensions (x,y,z)



Execution Node

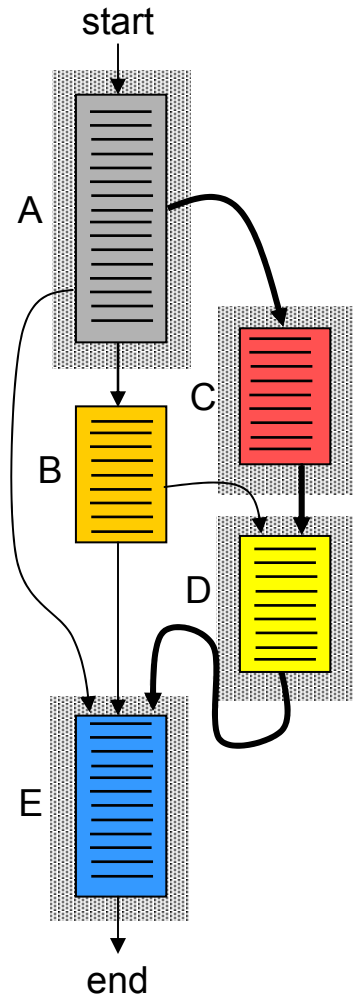


Instruction Buffers form  
a logical “z-dimension”  
in each node



4 logical *frames*  
each with 16 instruction slots

# Using *frames* for Speculation and ILP



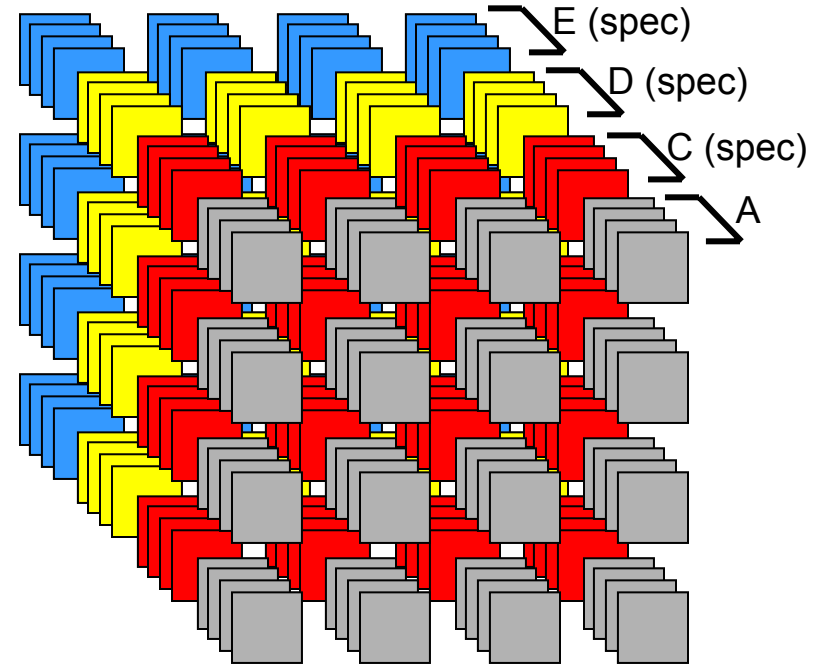
Map A onto GRID  
Start executing A

Predict C is next block  
Speculatively execute C

Predict is D is after C  
Speculatively execute D

Predict is E is after D  
Speculatively execute E

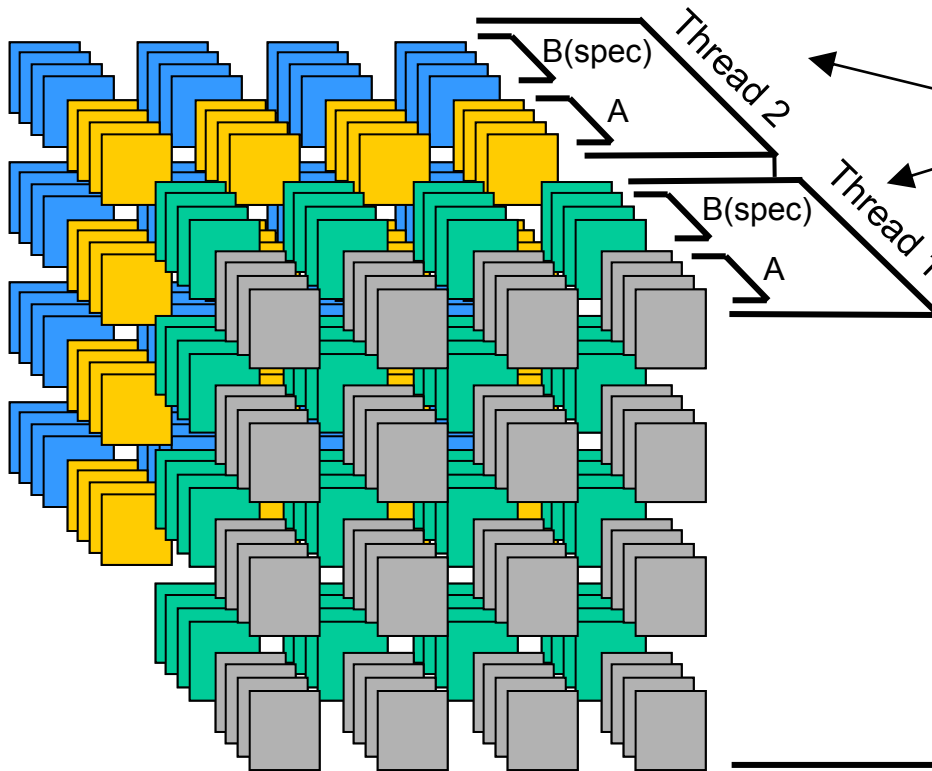
16 total frames (4 sets of 4)



## **Result:**

- Enormous effective instruction window for **extracting ILP**
- Increased utilization of execution units (*accuracy counts!*)
- Latency tolerance for GRID delays and Load instructions

# Using *frames* for TLP



Divide frame space among threads

- Each can be further divided to enable some degree of speculation
- Shown: 2 threads, each with 1 speculative block
- Alternate configuration might provide 4 threads

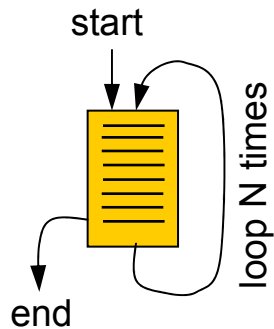
## **Result:**

- Simultaneous Multithreading (SMT) for Grid Processors
- Polymorphism: Use same resources in different ways for different workloads (“T-morph”)

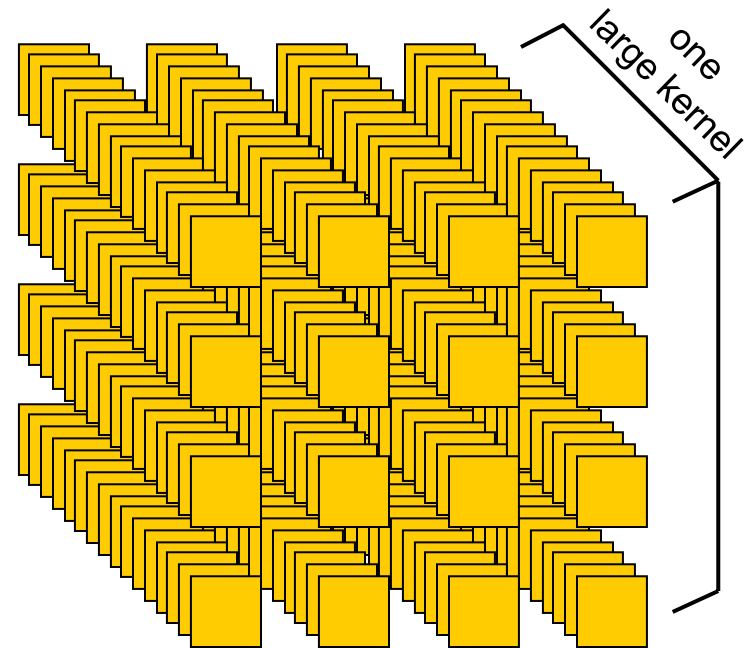
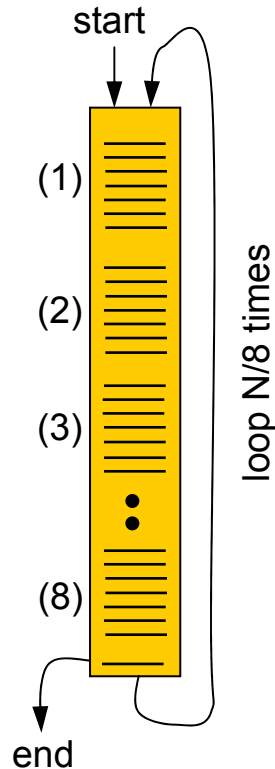
# Using *frames* for DLP

## Streaming Kernel:

- read input stream element
- process element
- write output stream element



unroll 8X



- Map very large blocks (kernels)
- Fetch once, use many times
- Not shown: **streaming data channels**

## **Result:**

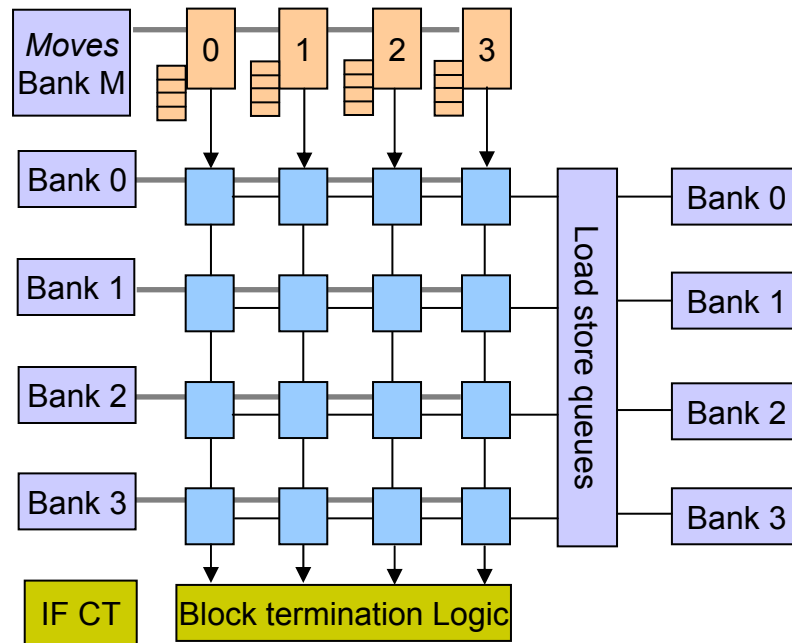
The instruction buffers act as a distributed I-Cache

Ability to absorb and process large amounts of streaming data

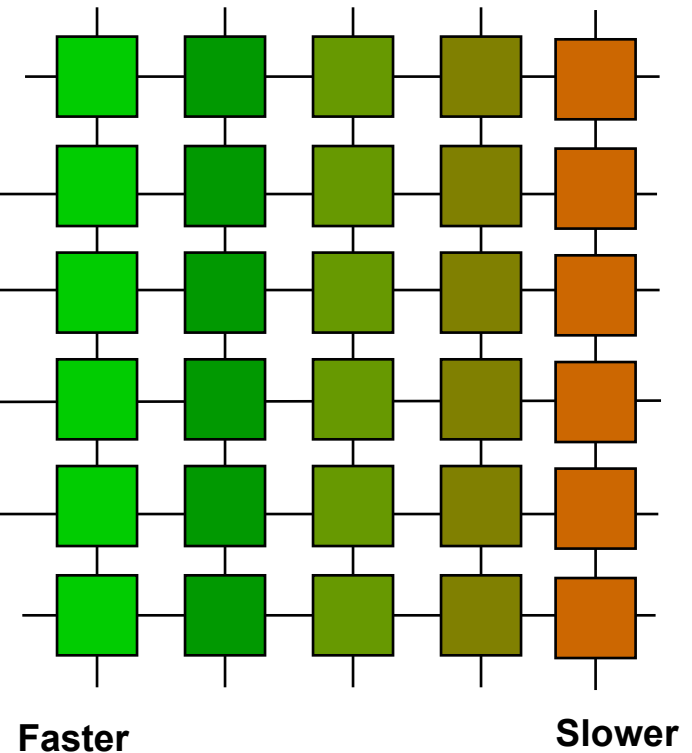
Another type of *Polymorphism* (“S-morph”)

# GPA and NUCA L2 Cache

## GRID Processor



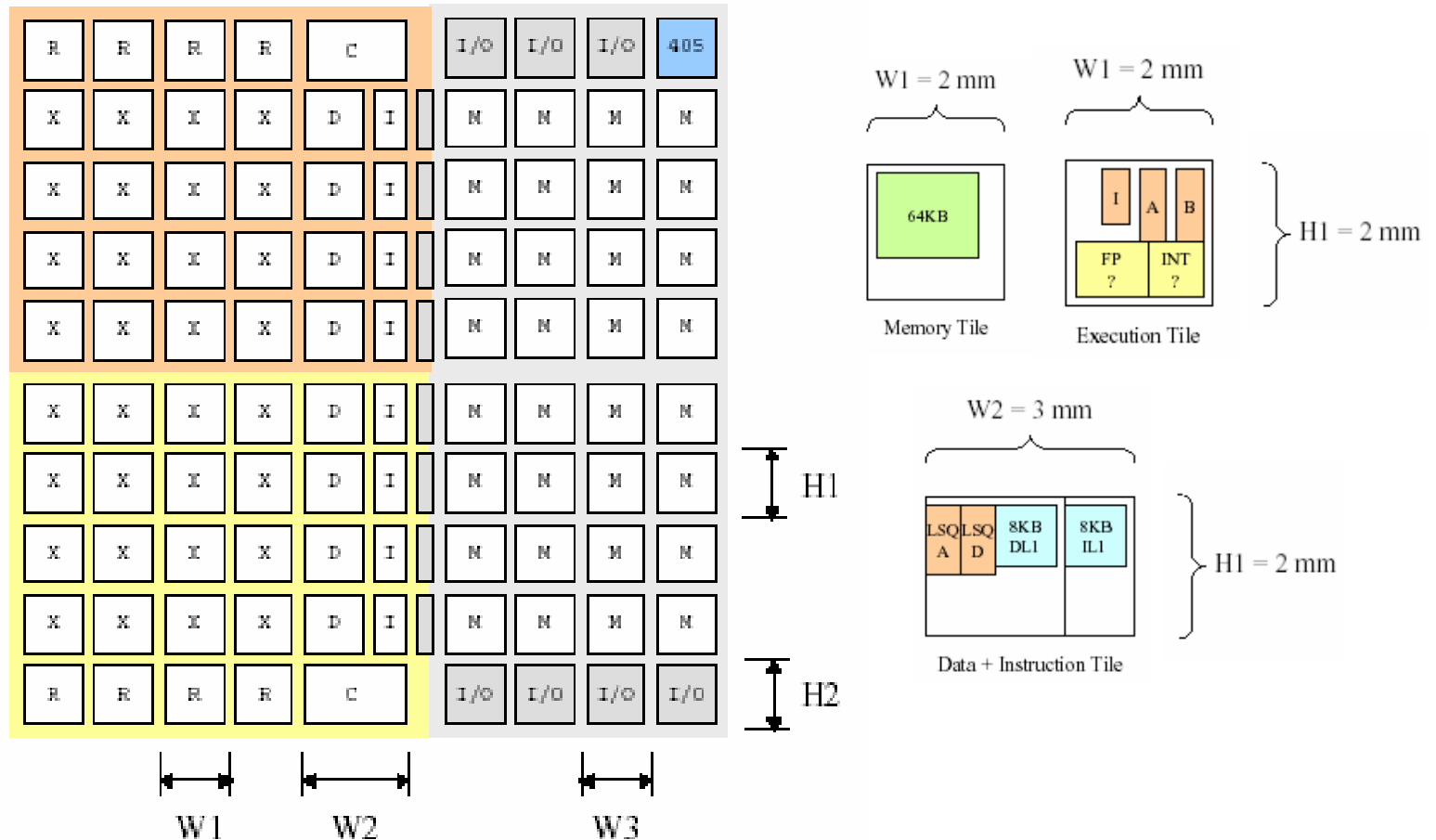
## L2 Cache Banks



**“NUCA”**

*Ref: Kim, etal, ASPLOS 2002*

# TRIPS Prototype – Tiled Floorplan Concept



Regular, pre-planned structure → helps manage [Wire Delay]  
→ improves [Designer Productivity]

# Concluding Comments

---

- Evolutionary design tends to accumulate complexity
- Revolutionary design is a tough sell, but allows a fresh start
- Transistor-effective design: *Transistors are not “free”*
  - Benefits: *incremental performance*
  - Direct costs: *complexity, power, area*
  - Indirect costs: *global infrastructure, deep sortability, yield*
- Getting the Attention of Industry:
  - Solve a ***new problem*** using an ***existing mechanism***
  - New mechanisms should contribute to solving multiple problems
    - Todd Austin’s DIVA is a good example
    - Emerging synergy between Reliability, Yield, PM, and Verification?
    - Find ways to move up the ***abstraction ladder***
  - Revolutionary ideas should offer at least 10X advantage
  - Technology transfer is hard ... *but, you must communicate!*

---

Fred Brooks on ***conceptual integrity***:

*“It is better to eliminate some frills and functionality to preserve the design ideals than to use many good, but independent and uncoordinated ideas”*

Questions?

## 2b: Superscalar Cores: *Key Circuit Elements*

---

	<b>Conventional 4-Issue</b>	<b>Hypothetical 16-issue</b>
<b>Execution</b>	2 FP, 2 INT, 2 LD/ST	8 FP, 8 INT, 8 LD/ST
<b>I-Cache</b>	64KB 1 Port, 64B (1 instance)	128KB <b>2 Ports</b> , 128B (1)
<b>Mapper</b>	8 port x 72-entry CAM (2)	<b>32 port x 512-entry CAM (2)</b>
<b>Issue Queue</b>	4P x 20-entry dual CAM (3)	4P x <b>40-entry dual CAM (12)</b>
<b>RegFiles</b>	72-entry, 4R, 5W ports (4)	<b>512-entry, 4R, 18W ports (8)</b>
<b>D-Cache</b>	32KB 2R/1W ports (1)	128KB <b>8R/4W ports (1)</b>

*... what happens if we partition these to run at 8 FO4?*

*... what about these?*