A System-level Perspective on Adaptive & Integrated Memory Systems

Rajesh K. Gupta

University of California, San Diego

http://nvsl.ucsd.edu

In collaboration with Steve Swanson

Non-volatile Systems Workshop, UCSD

Points I want to make today

1. Microelectronic realities are a driver of change in memory systems
   - Always driven by cost (scalability)

2. Programming has always been an unmet challenge, it only gets better

3. New generation of machines will see memory systems that are truly lightweight and flexible, courtesy meta-data handling methods from the Web.

Outline

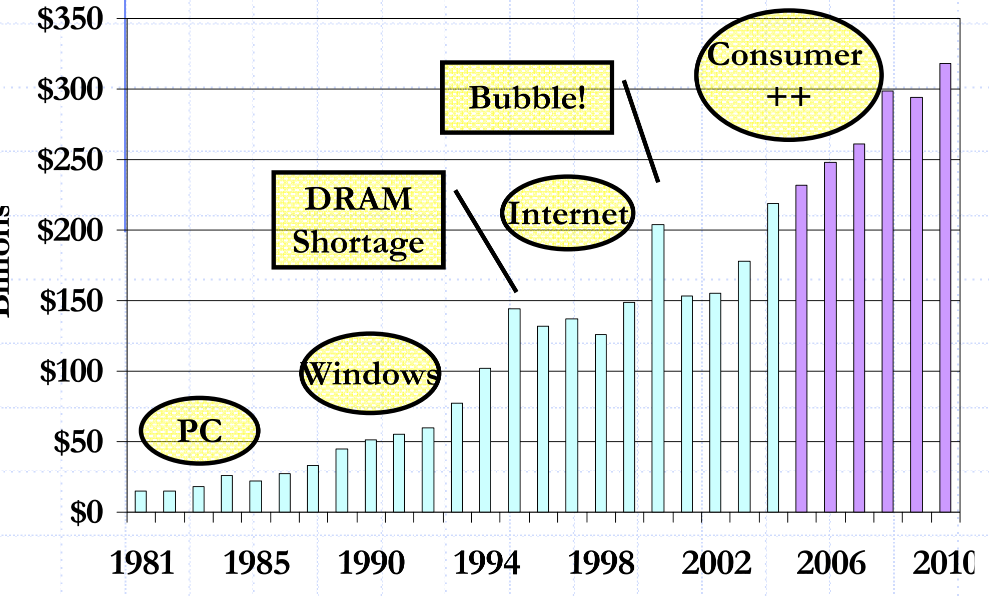
- The Technology Drive and Drivers
  - 1990’s: The Era of Interconnect
  - 2000’s: The Era of Power
  - 2010’s: The Era of New Materials, Uncertainty
- 2010’s: Rise of Universal Memory
  - Storage abstractions
  - Programming, Persistence and NVM.

Projects & Credits:

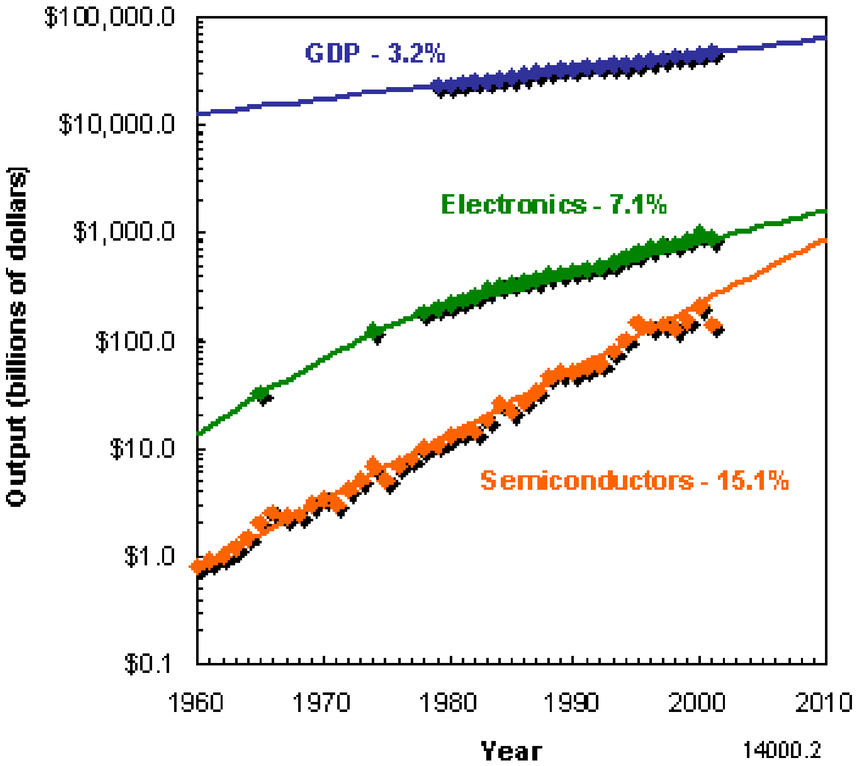
- Adaptive Memory Reconfiguration Management (AMRM) Project, DARPA/ITO
- MORPH Machines, NSF/HPCC
  - Alex Nicolau, Andrew Chien, Alex Veidenbaum, Brad Calder
  - Ali Dasdan, Weiyu Tang, Xiaomei Ji, Paolo D'Alberto, Dan Nicolaescu
  - Rajesh Satapathy, Vivek Sinha, Piyush Garg, Prashant Arora, Chun Chang.
- Non-Volatile Systems Laboratory
  - Steve Swanson
  - Amin Akel, Joel Coburn, Arup De, Adrian Caufield, Laura Grupp, Michael Wei.

Process & Device Technology Drivers

- $250B industry driving $1T systems industry
  - 30-year impact on GDP is over 21% todav!

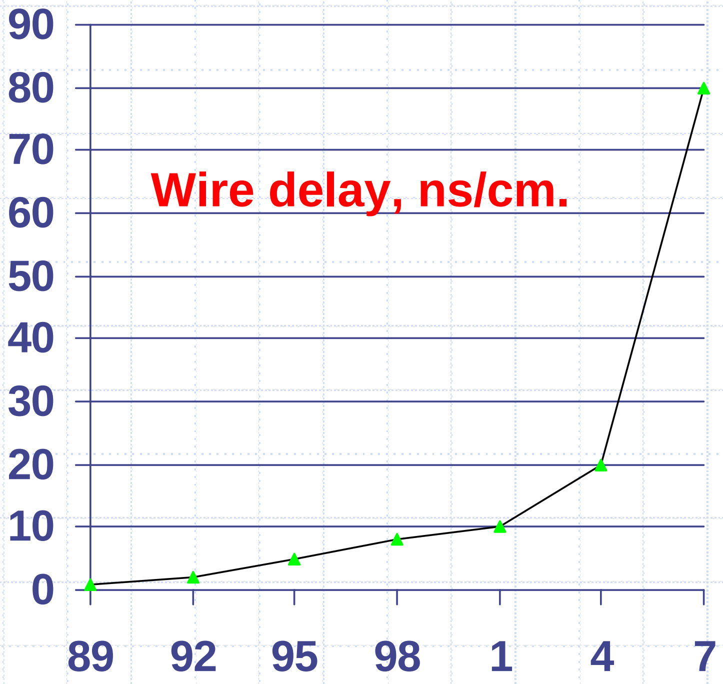


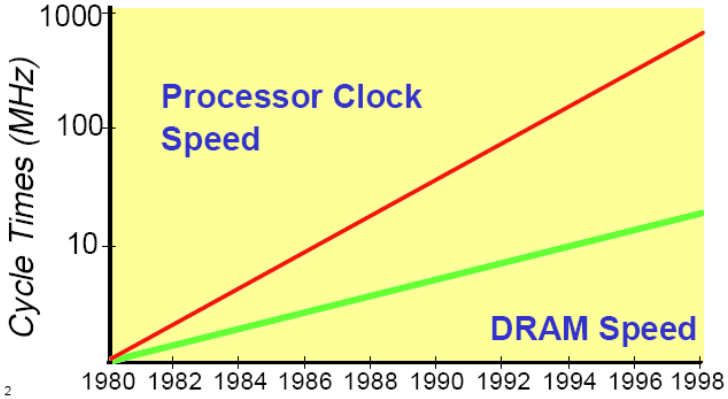
Source: Gartner (Includes Optics)



- Technology innovations must face material, process, and device realities.

1990’s: Interconnect on our minds



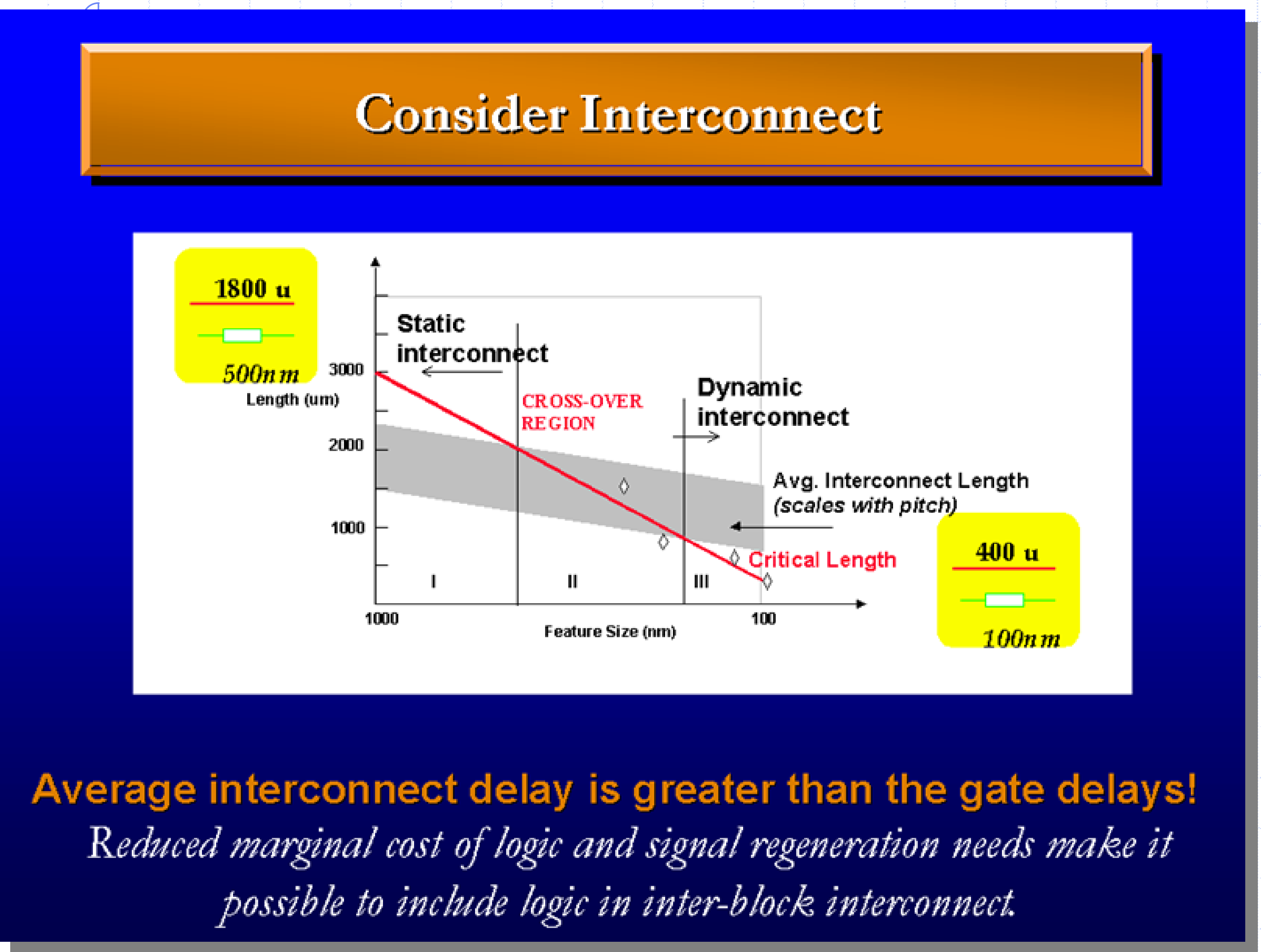


At 4 GHz, light travels 75mm in a clock cycle.

In 6ps, light travels about 2mm.

- Logic versus Memory tension
- Locality assumptions under attack
- Memory Wall

90’s: The Era of Interconnect

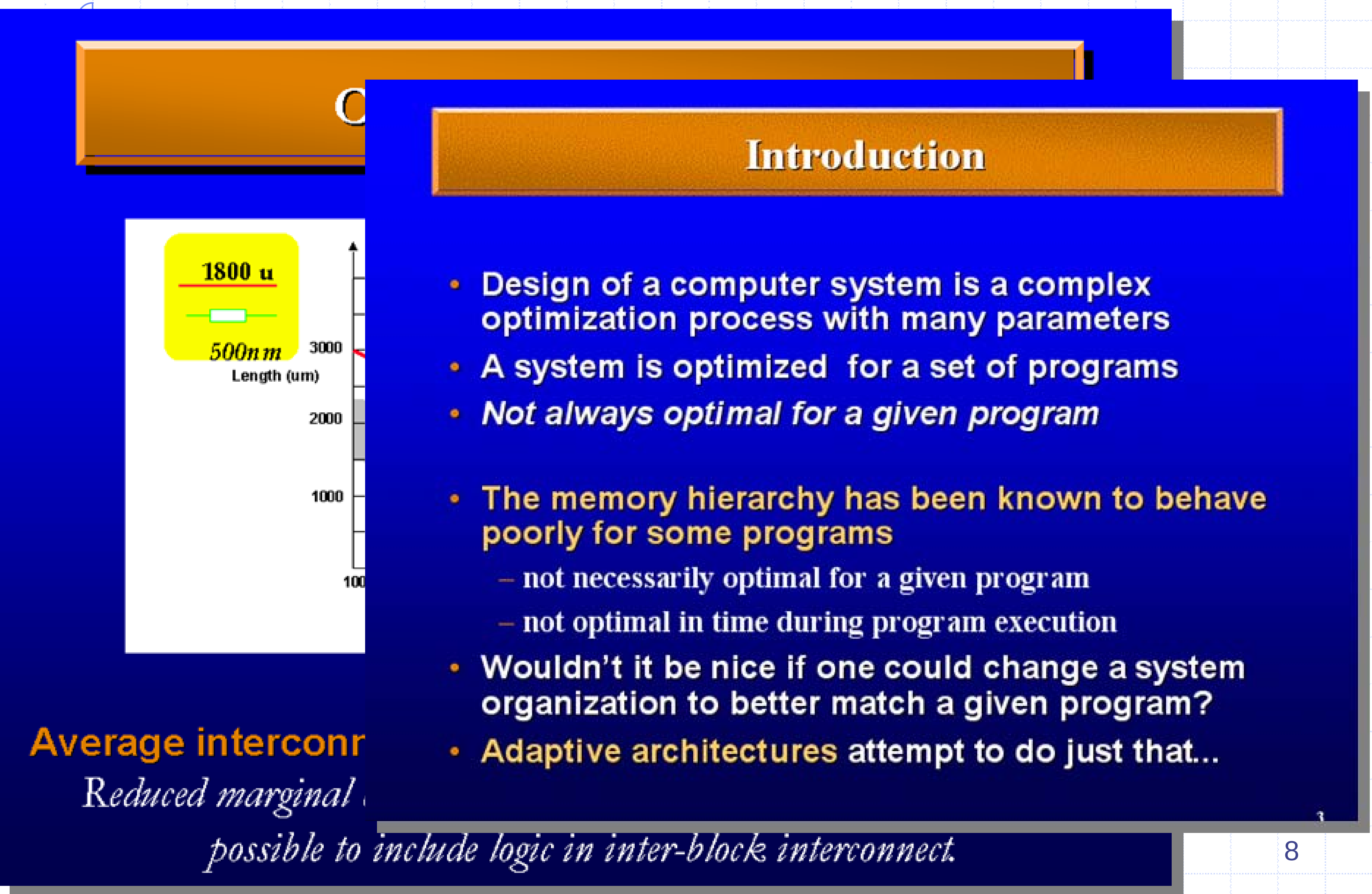


Consider Interconnect

Average interconnect delay is greater than the gate delays!

Reduced marginal cost of logic and signal regeneration needs make it possible to include logic in inter-block interconnect.

90's: The Era of Interconnect



Average interconn

Reduced marginal

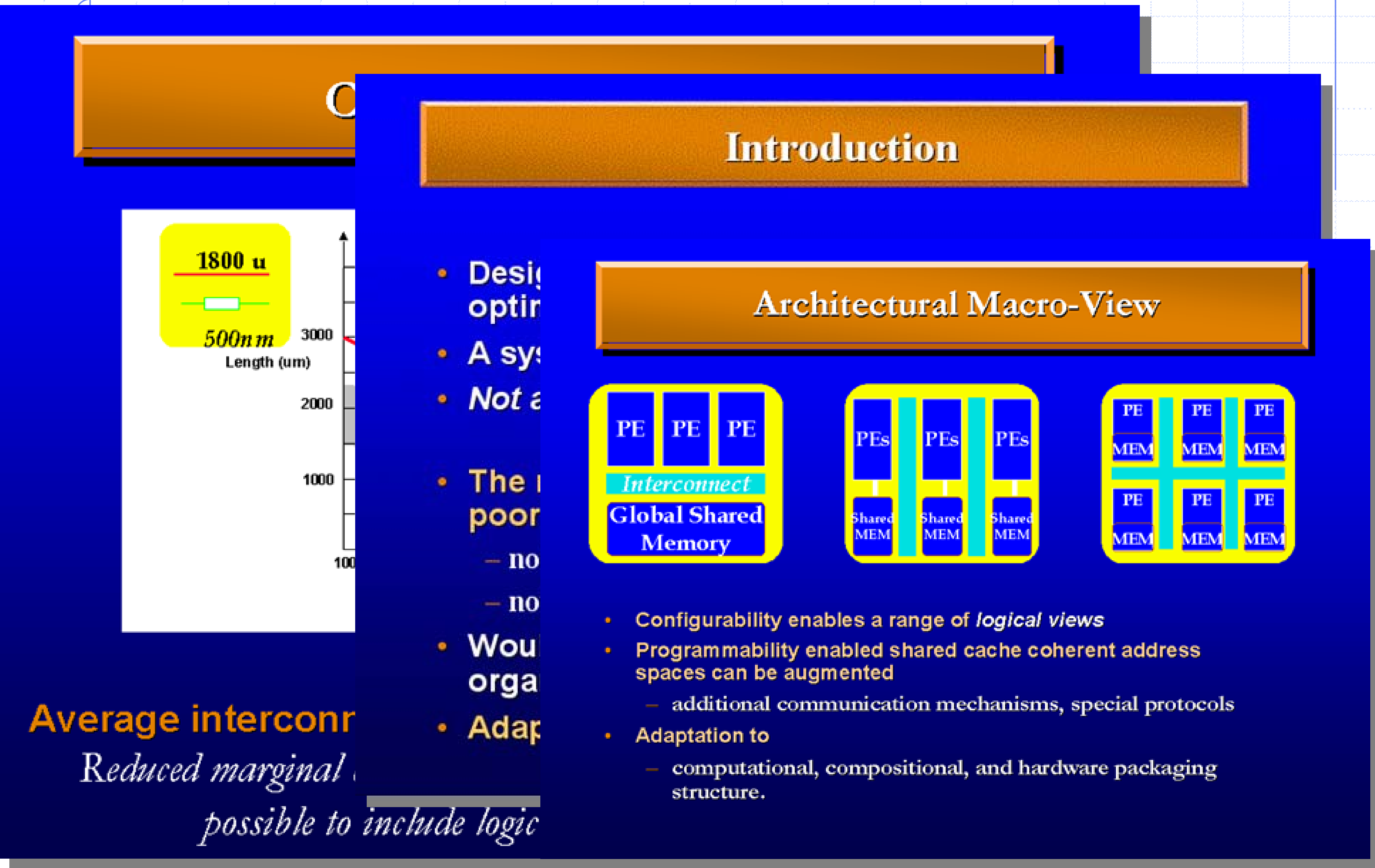
possible to include logic in inter-block interconnect.

Introduction

- Design of a computer system is a complex optimization process with many parameters
- A system is optimized for a set of programs
- *Not always optimal for a given program*
- The memory hierarchy has been known to behave poorly for some programs
  - not necessarily optimal for a given program
  - not optimal in time during program execution
- Wouldn't it be nice if one could change a system organization to better match a given program?
- Adaptive architectures attempt to do just that...

3

90’s: The Era of Interconnect



O

Average interconn

Reduced marginal

possible to include logic

Introduction

- Desi optir
- A sys
- *Not a*
- The r poor
  - no
  - no
- Wou orga
- Adap

Architectural Macro-View

- Configurability enables a range of *logical views*
- Programmability enabled shared cache coherent address spaces can be augmented
  - additional communication mechanisms, special protocols
- Adaptation to
  - computational, compositional, and hardware packaging structure.

Architectural Adaptation

- **Each of the following elements can benefit from increased adaptability**

  (above and beyond CPU programming)
  - **CPU**
  - **Memory hierarchy : eliminate false sharing**
  - **Memory system : virtual memory layout based on cache miss data**
  - **IO : disk layout based on access pattern**
  - **Network interface : scheduling to reduced end-to-end latency**
- **Adaptability used to build**
  - **programmable engines in IO, memory controllers, cache controllers, network devices**
  - **configurable data-paths and logic in any part of the system**
  - **configurable queueing in scheduling for interconnect, devices, memory**
  - **smart interfaces for information flow from applications to hardware**
  - **performance monitoring and coordinated resource management...**

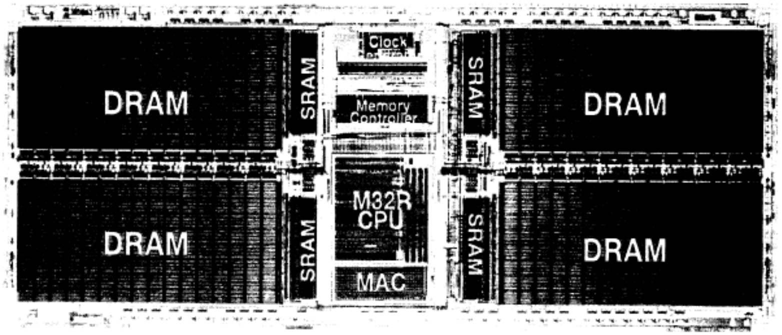
Intelligent interfaces, information formats, mechanisms and policies.

The Vision

- Gates: zero, Interconnect: infinity

4 MB DRAM/M32R accessed as memory or CPU.

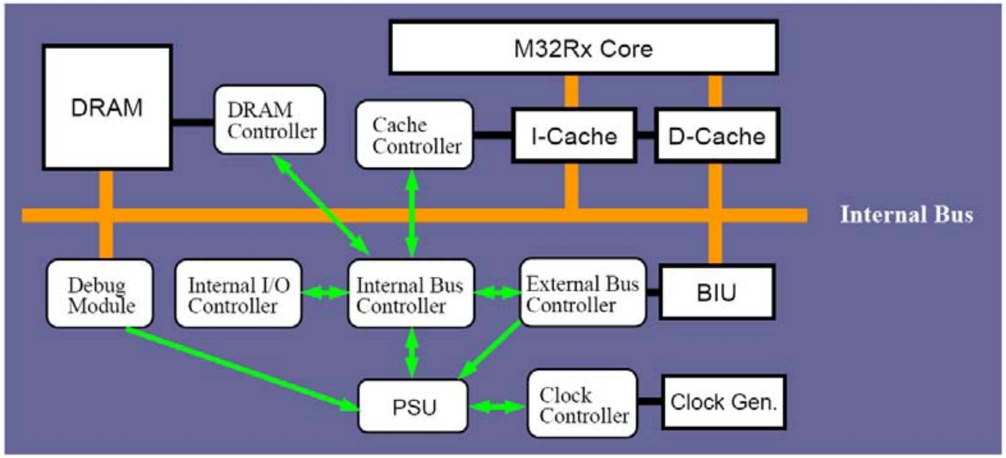
M32R/DRAM



8

MITSUBISHI E

0.25micron, M3, 100 sq mm, 100 MHz



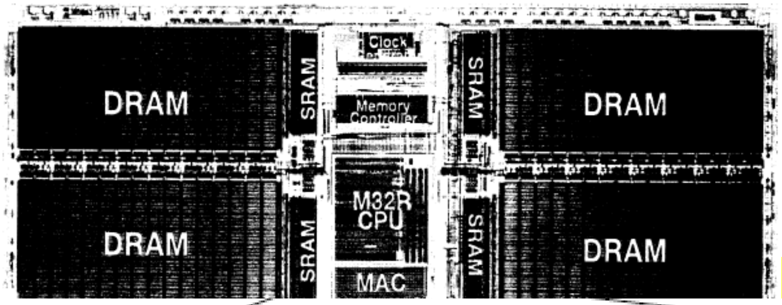
PSU: Power Saving Unit

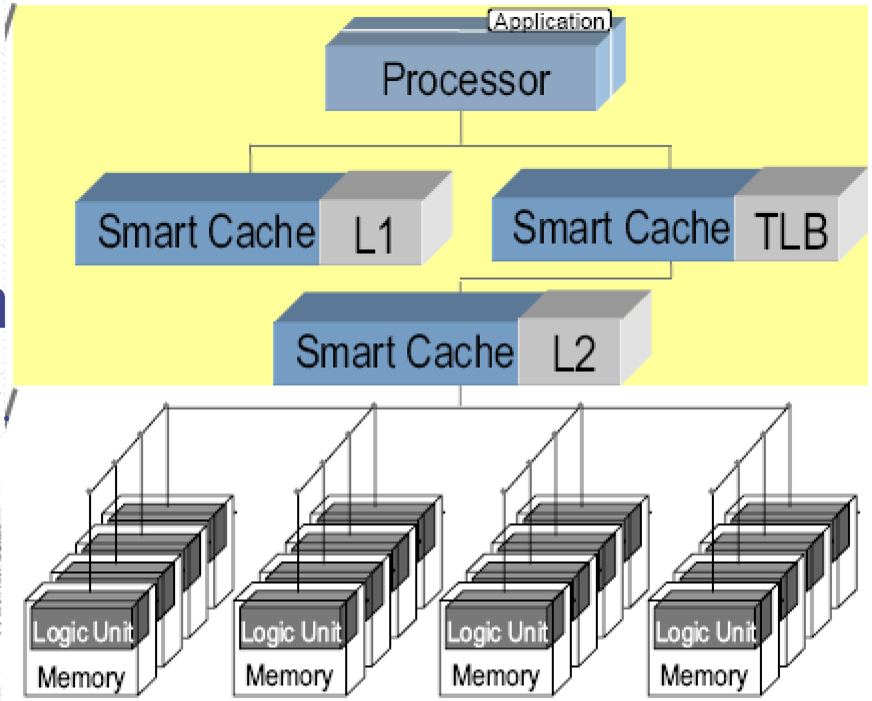
The Vision

- Gates: zero, Intercon

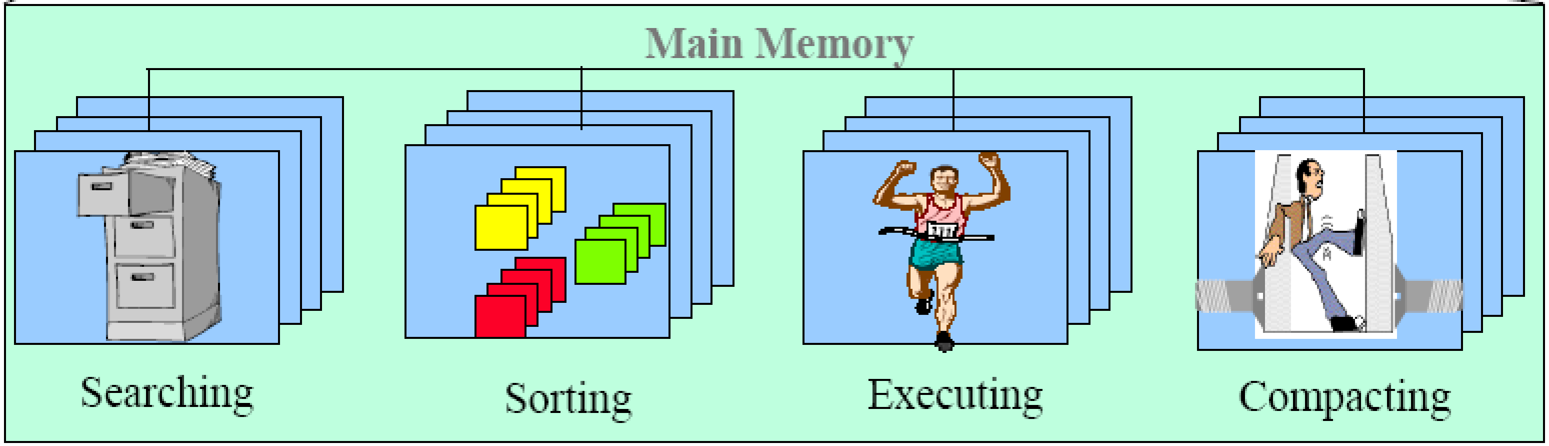
4 MB DRAM/M32R accessed as memory or CPU/

M32R/DRAM



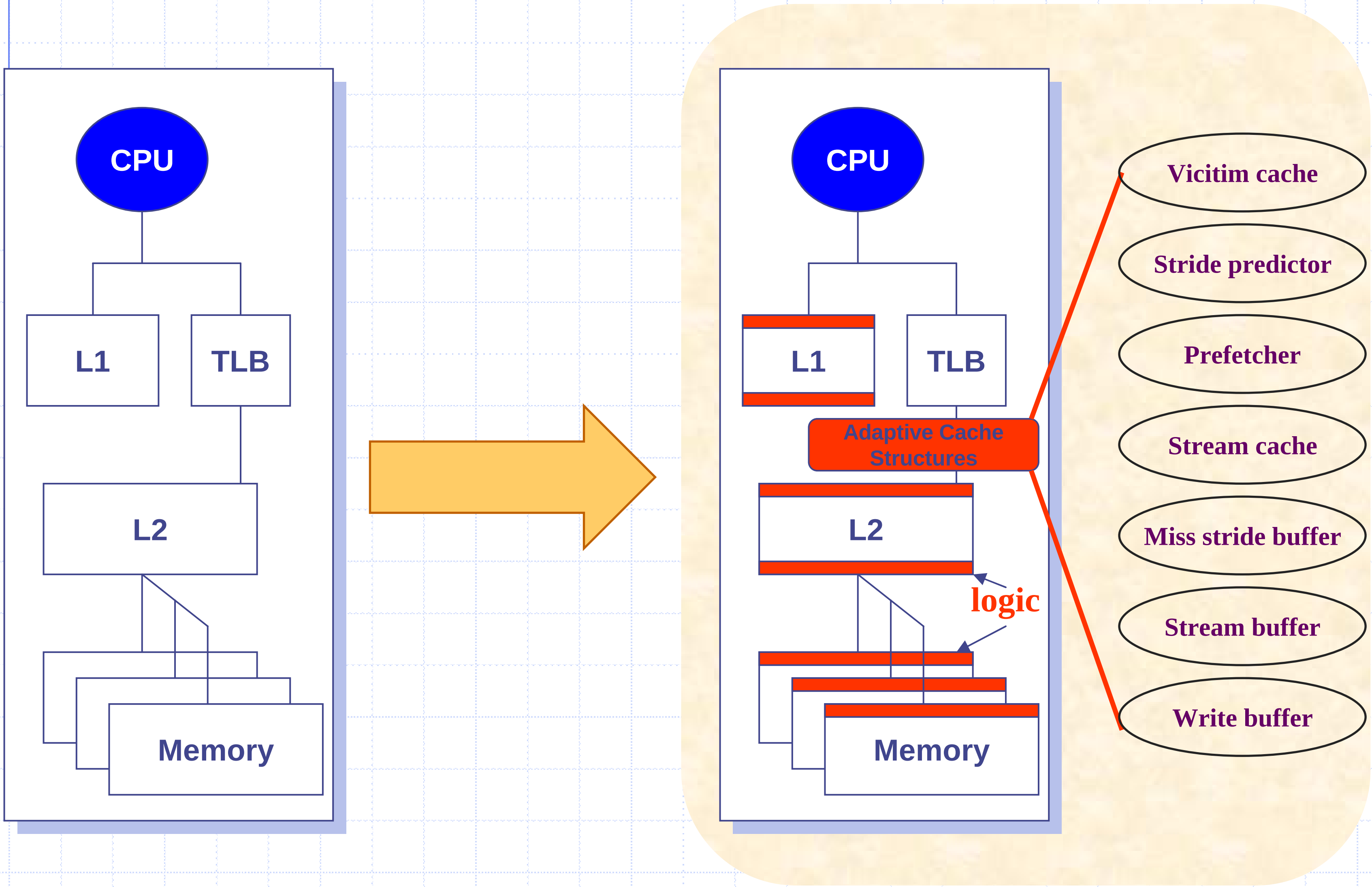


Leverage application knowledge and run-
om
erns

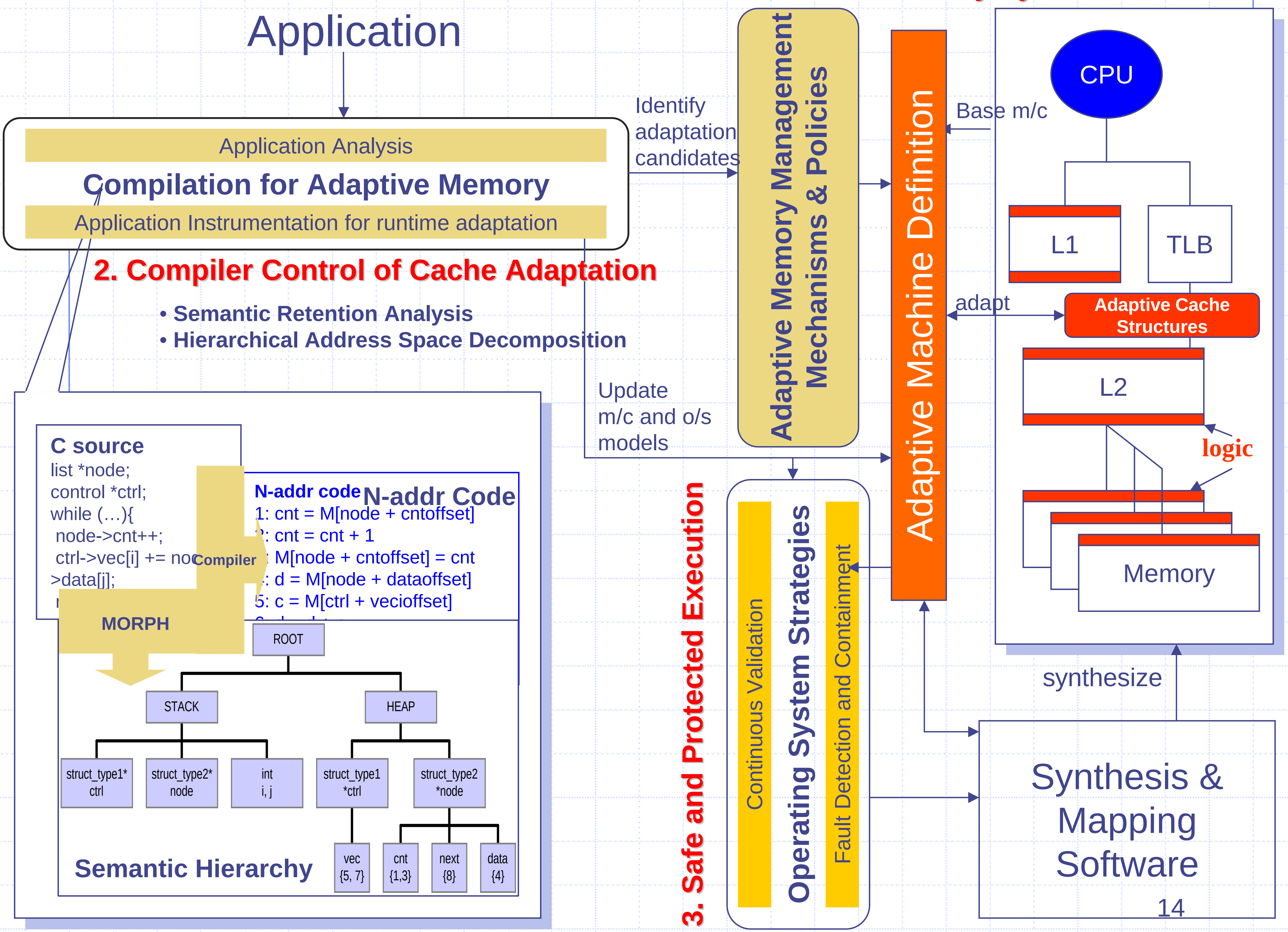


Approved for Public Release, Distribution Unlimited

Develop an adaptive cache memory architecture that achieves peak system efficiency by enabling applications to optimally manage movement of data through the memory hierarchy.



1. Flexible Memory System Architecture



Application

Application Analysis

Compilation for Adaptive Memory

Application Instrumentation for runtime adaptation

2. Compiler Control of Cache Adaptation

- **Semantic Retention Analysis**
- **Hierarchical Address Space Decomposition**

Identify adaptation candidates

Update m/c and o/s models

Adaptive Memory Management Mechanisms & Policies

Adaptive Machine Definition

Base m/c

adapt

C source

```
list *node;
control *ctrl;
while (…){
 node->cnt++;
 ctrl->vec[i] += no
>data[j];
```

N-addr code **N-addr Code**

```
1: cnt = M[node + cntoffset]
?: cnt = cnt + 1
: M[node + cntoffset] = cnt
: d = M[node + dataoffset]
5: c = M[ctrl + vecioffset]
```

Semantic Hierarchy

3. Safe and Protected Execution

Operating System Strategies

Continuous Validation

Fault Detection and Containment

synthesize

Synthesis & Mapping Software

1. Architectural Experiments

A Cache assists

- choice of mechanisms & mechanism “parameters” (size, lookahead etc.)
  - » **Prefetching**
    - Stream Buffers
    - Stride-directed: based on address alone
    - Time Stride: prefetch the same address using the number of intervening misses as lookahead
    - Pointer Stride
  - » **Victim Cache, Write Buffer**

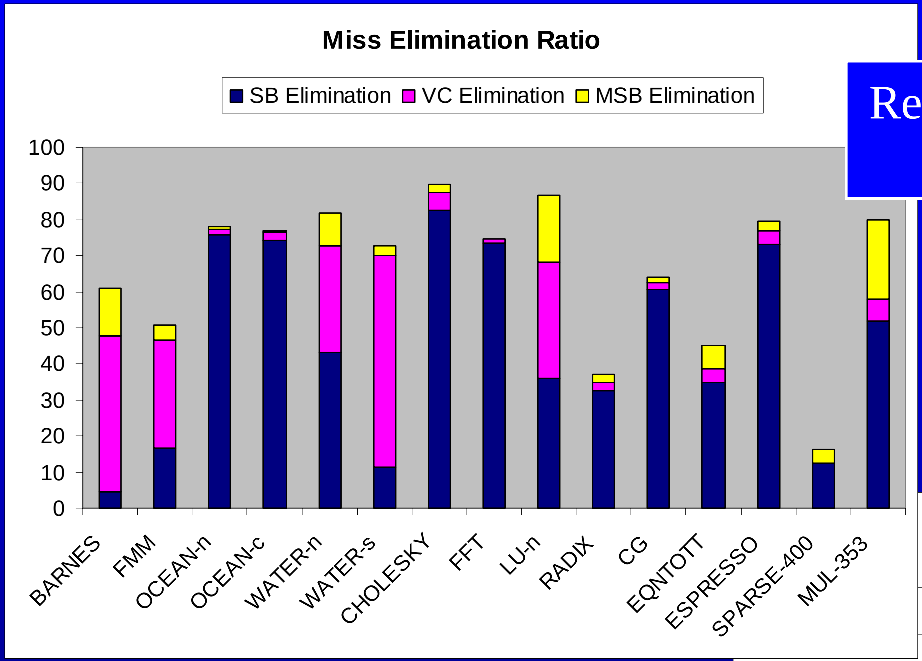
B Adaptive cache organizations

- line size, fetch size, policies, adaptivity algorithms

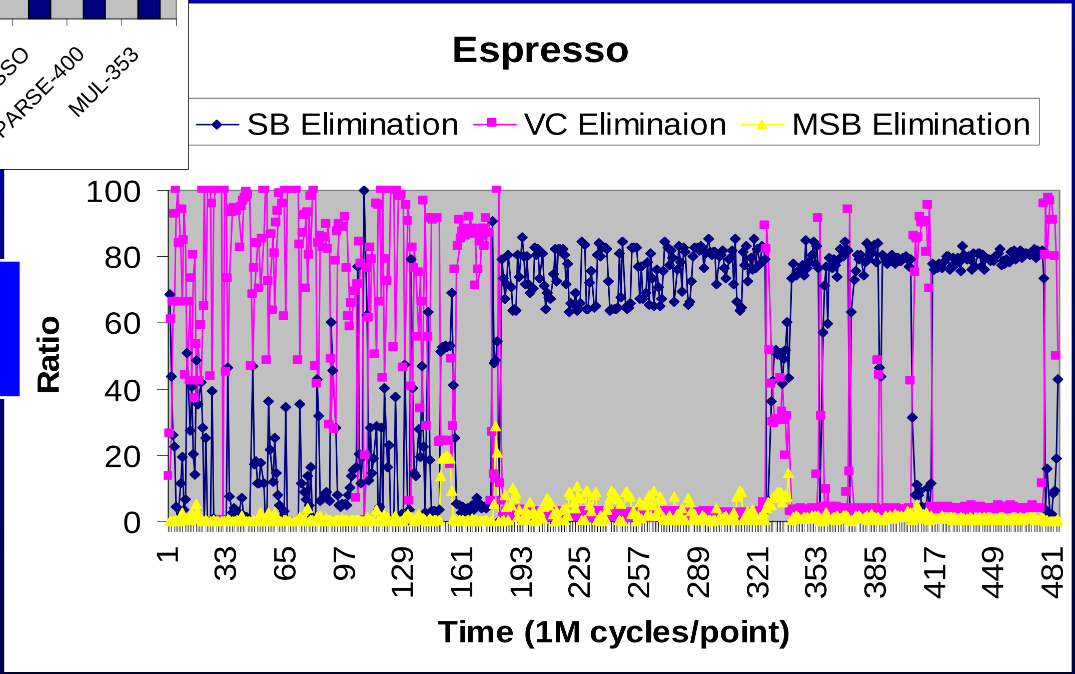
C Hierarchy organization

- what are the datapaths like?
- how much parallelism is possible in the hierarchy?
- coordination between adaptations at multiple levels.

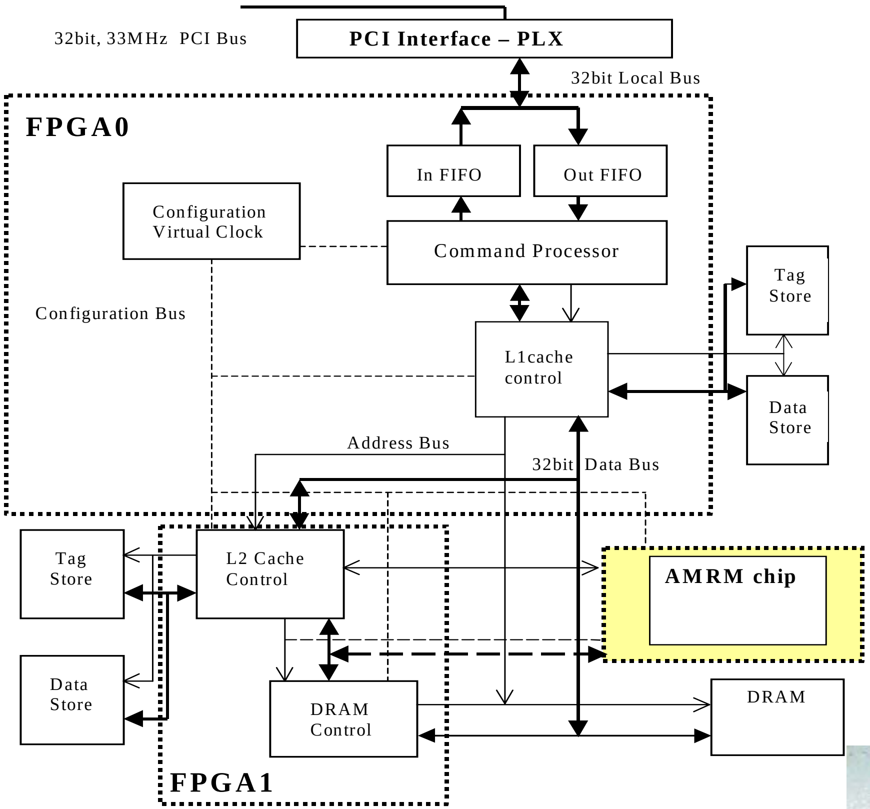
A. Effect of Cache Assists

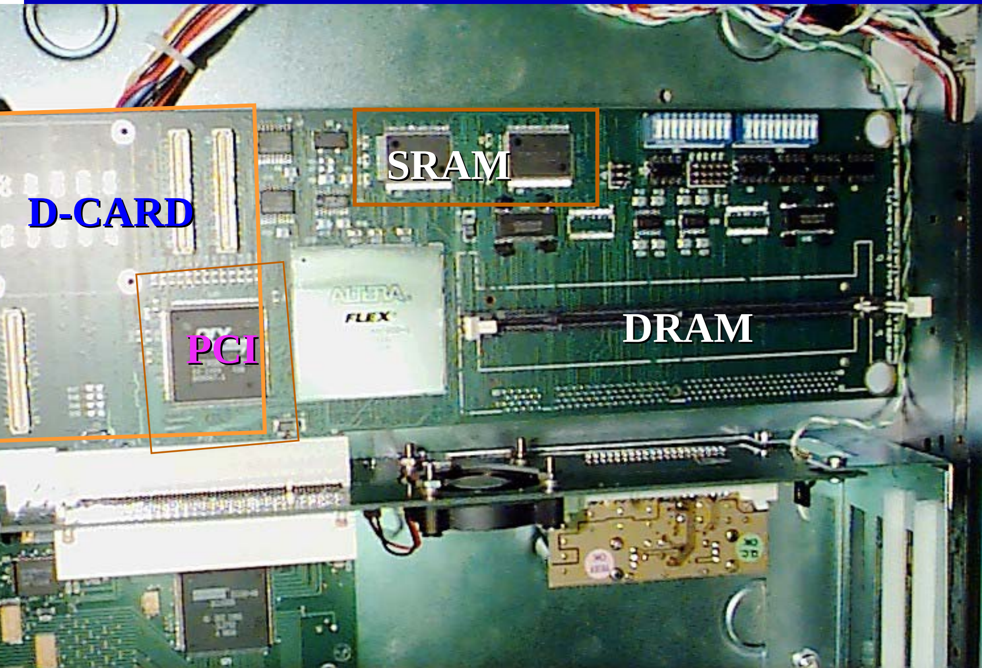


Reduction in miss rate due to static choice of appropriate assists

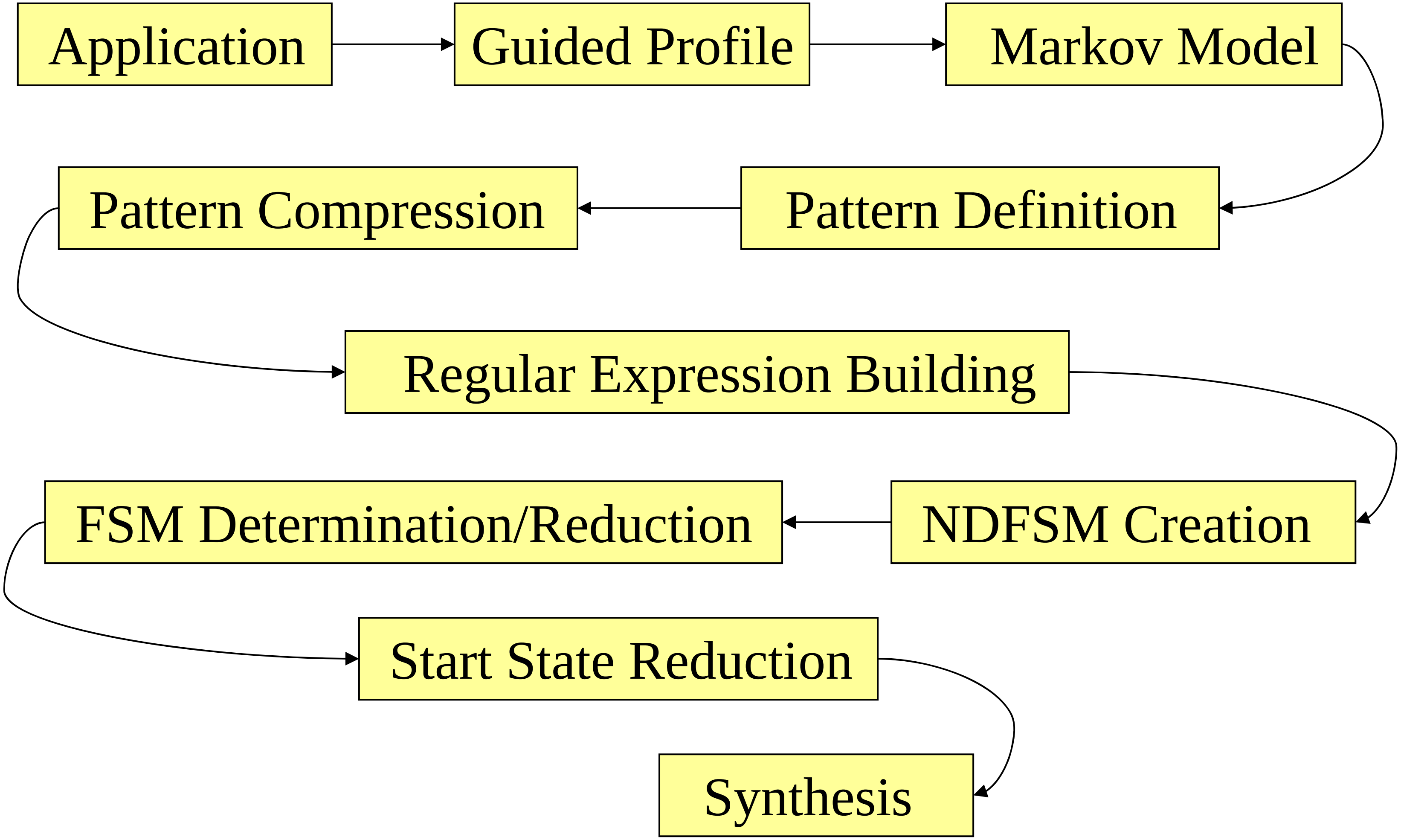


Reduction in miss rate due to dynamic choice of appropriate assists


32bit, 33MHz PCI Bus
PCI Interface – PLX
32bit Local Bus
FPGA0
In FIFO
Out FIFO
Configuration Virtual Clock
Command Processor
Tag Store
Configuration Bus
L1cache control
Data Store
Address Bus
32bit Data Bus
Tag Store
L2 Cache Control
AMRM chip
Data Store
DRAM Control
DRAM
FPGA1


SRAM
D-CARD
PCI
DRAM

Automated Design Overview



Application

Guided Profile

Markov Model

Pattern Definition

Pattern Compression

Regular Expression Building

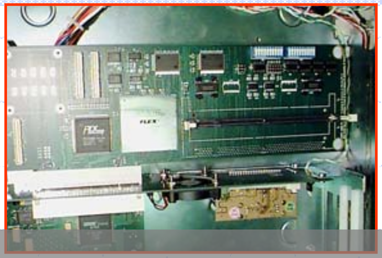
NDFSM Creation

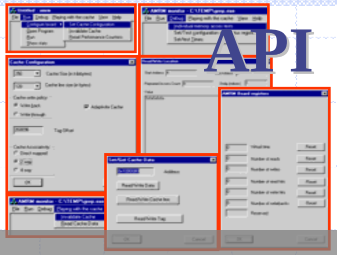
FSM Determination/Reduction

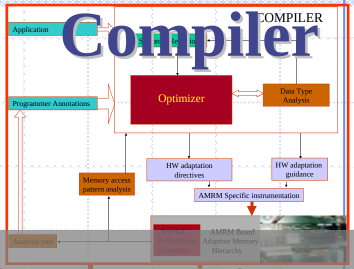
Start State Reduction

Synthesis

AMRM Results



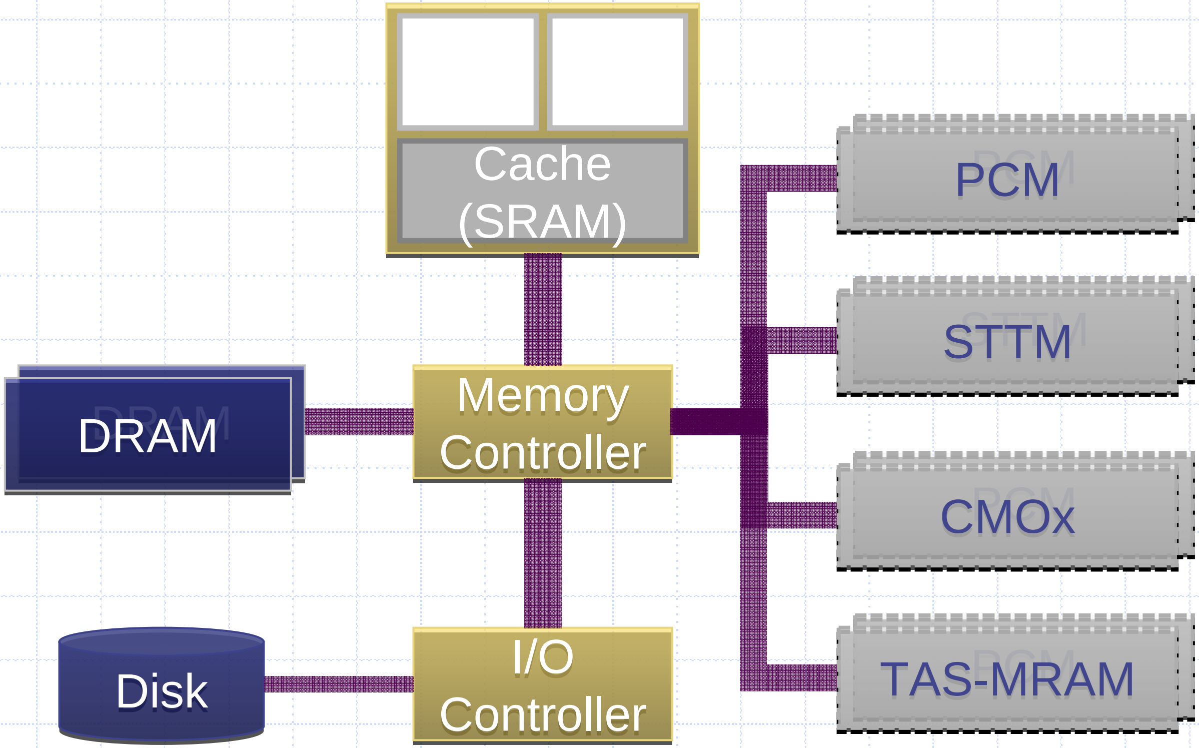




- Extensive exploration of the space of adaptive architectural assists, adaptive line and fetch sizes
  - targeted architectural assists for conflict, capacity and compulsory misses
  - MRR of 64% (CG), 50% (FMM), 80% (MM), upto 90% reduction
  - 20% miss-rate reduction even over optimal static configurations and upto 50% reduction in memory traffic
  - 12.9X speedup on NAS-CG [4.33 (L1/L2) X 1.5 (Compiler) X 2-3(HW)]
- Evaluation platform and prototypes
  - Adaptive cache simulation workbench with accurate SS/OO processor models and fast simulation through "execution sampling" (0.5M ips)
  - Fully functional hardware board available for experimentation with adaptive memory controllers with built-in multiple architectural modes; and a library of synthesizable memory and cache controllers
- Compiler control of adaptation
  - enable application analysis; semantic retention and direct reconfiguration
  - ADL as interface between architectural adaptivity and its software control
  - automatic loop instrumentation; API to AMRM hardware
  - Safety and protection guarantees: a framework for safe reconfiguration.

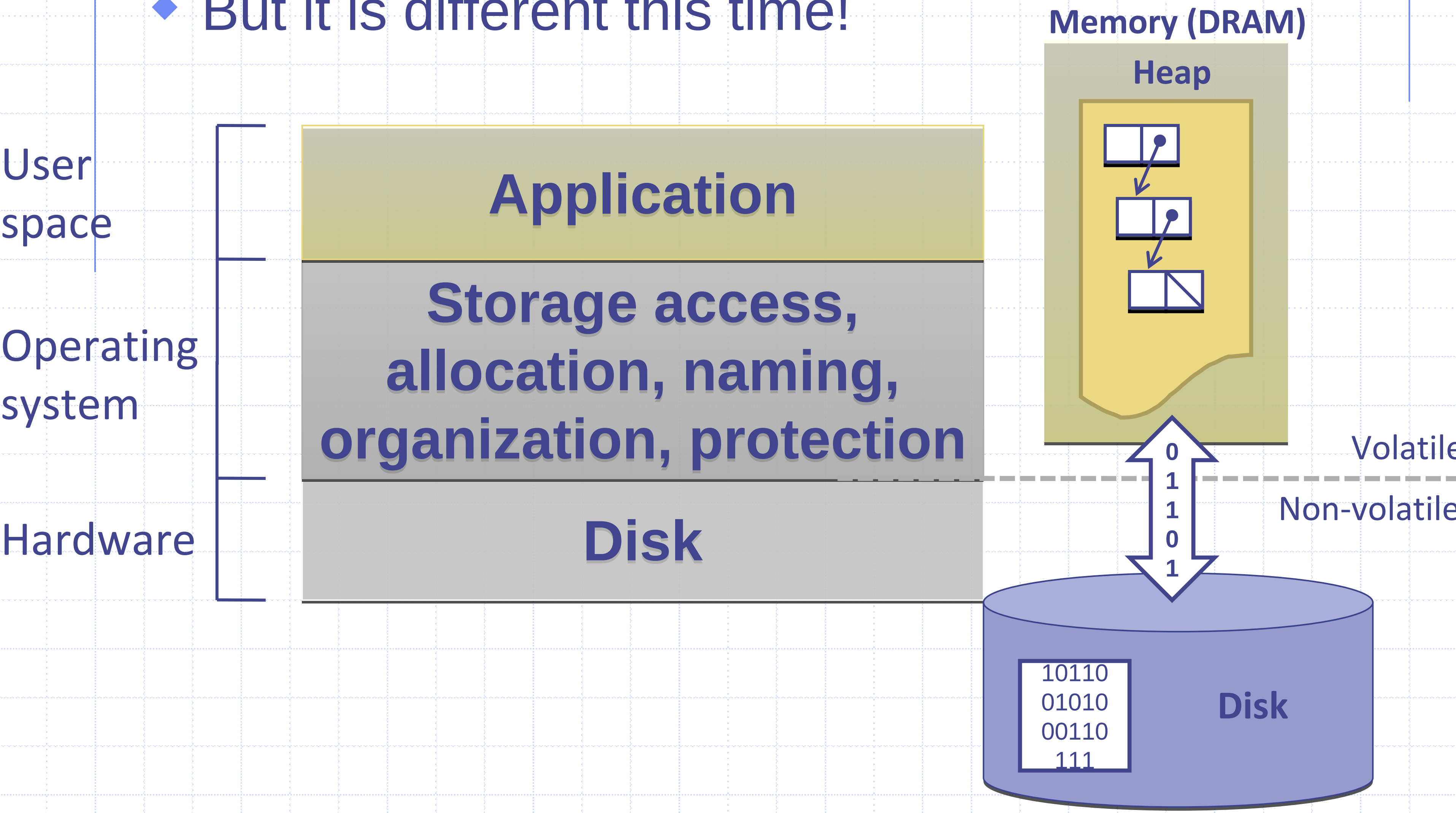
2010’s: New Materials, Exotic Devices

- Once again, memory is the process lead
- Explosion in memory technologies
  - All solid state, almost all NVM
  - Multi-layer, multi-bit almost a given (cf: variability)



Disk Has Been Under Attack Before

- But it is different this time!



User space

Operating system

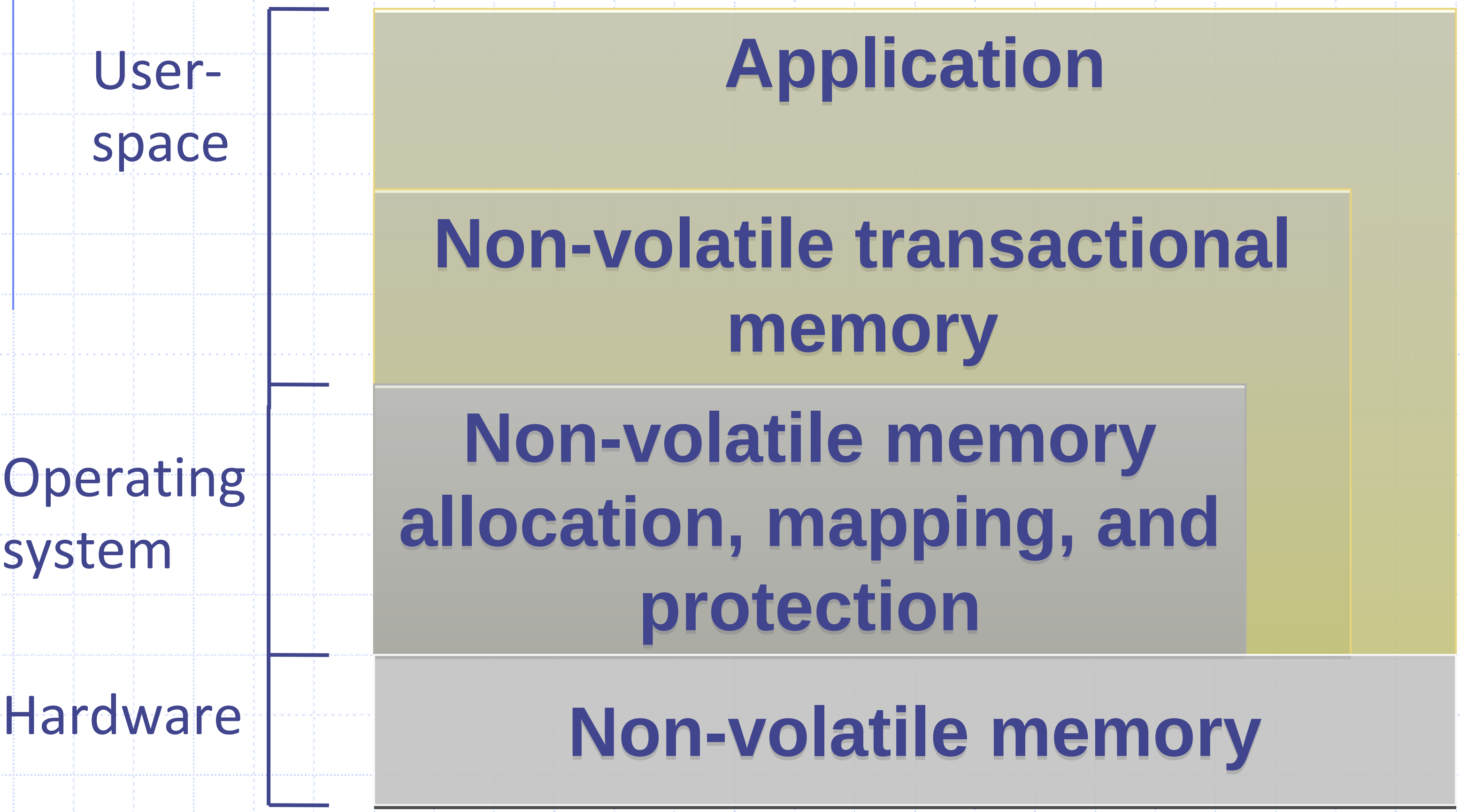
Hardware

Application

Storage access, allocation, naming, organization, protection

Disk

NVTM: Systematic OS-Bypass



User-space

Operating system

Hardware

Application

Non-volatile transactional memory

Non-volatile memory allocation, mapping, and protection

Non-volatile memory

Real Challenge: Programming

- Programming Model:
  - An abstraction of the computing machine and/or its environment that the programmer needs to know in order to effectively use machine.
  - Related to “Model of Computation” (MOC)

- A significant PM change can be a precursor of major turn in PL
  - PL evolution often emerges in the form of new libraries or APIs, language extension and analyses.

- Two drivers for this change:
  1. Rich and diverse computing fabrics (in silicon): memory like
  2. Growing gap in abstractions at various layers: on how data is seen

Abstraction Gaps or Mismatches

- Persistence is of fundamental importance to every application
  - Data outlives application
  - Traditionally such data comes from a database
  - Yet, programs don’t know about these.
- Growing gaps
  - Persistent storage access through declarative means
  - Volatile storage accessed imperatively, object orientation
- Managing mismatches
  - Java/Relational, XML/Relational

Supporting Persistence

- What is the support for creating, maintaining transient versus persistent objects in a PL?
  - Tables in SQL versus Classes in Java
  - Classes work well with design patterns in programming but then how does one map operations on classes (inheritance, polymorphism) to tabular operations?
  - 30% of Java app code in business applications is written to do this bridging
  - One common method is to use serialization in Java: takes a snapshot of the objects and writes out a byte stream (to a file or to a DB)
- Object versus Relational
  - Solved with persistence layers in object-oriented applications (ORMs such as Java Persistence or Hibernate)
  - Mediates applications interaction with a relational DB
- ORMs work by meta data
  - Recent advances in semi-structured data, meta-data methods, reflection and introspection

Emerging Capabilities in Programming: Meta Data Handling

- Self-documenting/extensible “tags”
  - Nested tags enable exchange of data (not just documents)
  - Data interchange is the basis for integration of services on the web
- There is no distinction between data and schema
  - Simplest XML is a labeled ordered tree with labels on nodes, and possible data values at the leaves.
- Schema extracted through Data Type Definitions (DTDs)
  - Not quite PL data type:
    - Values are not constrained (all values as strin difficult; Inability to separate type of an eleme
- New flexible/extensible types and schemas
  - E.g., regular expressions over trees
  - Ability to talk “about” data / queries through r

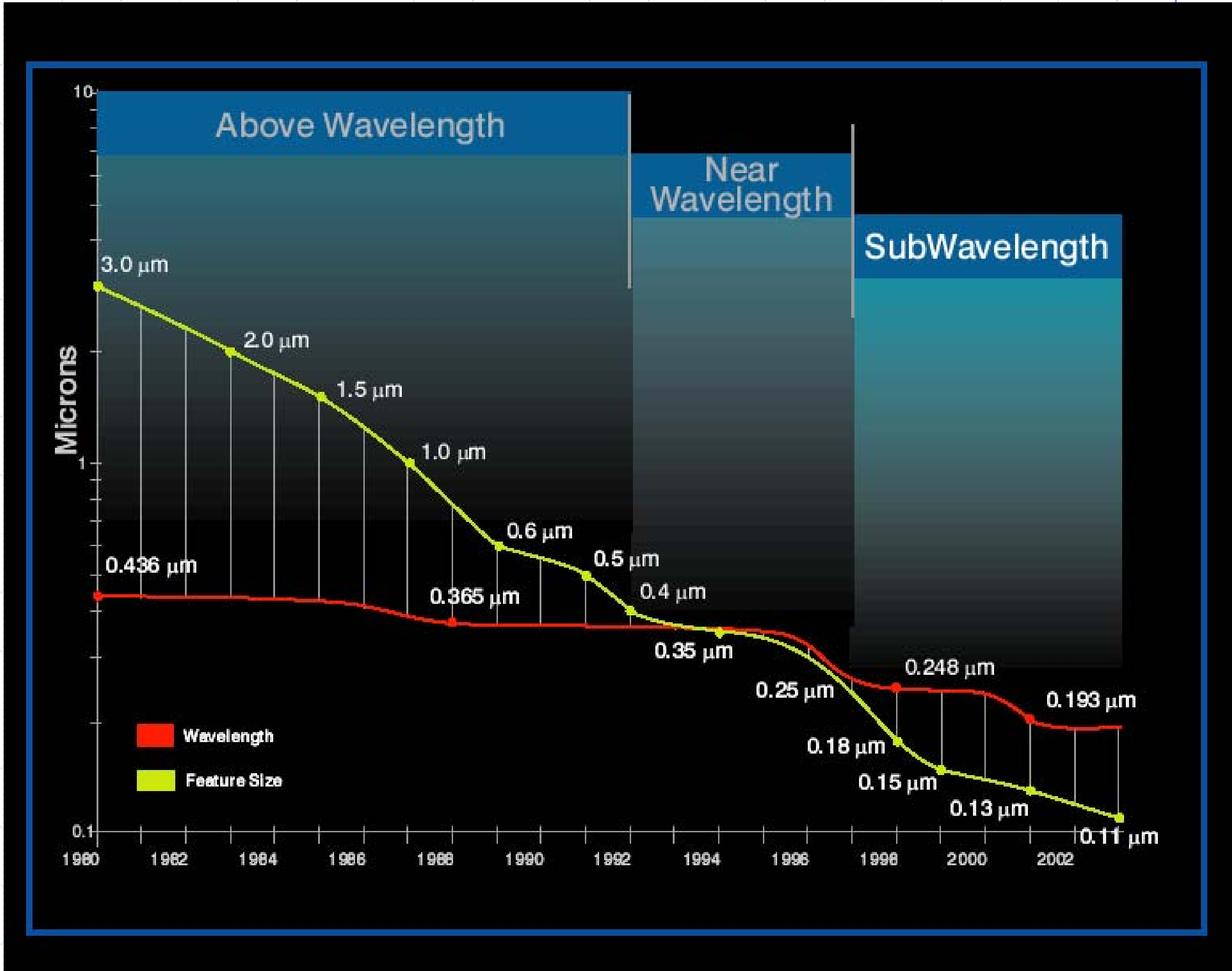
Example

```
<dealer> <UsedCars> <ad>
<model>Honda</model><yr>92</yr>
</ad></UsedCars>
<NewCars> <ad>
<model>Prius</model></ad></NewCars
> </dealer>
```

root: dealer
dealer → UsedCars, NewCars
UsedCars → ad*
NewCars → ad*
ad → modelyear | model

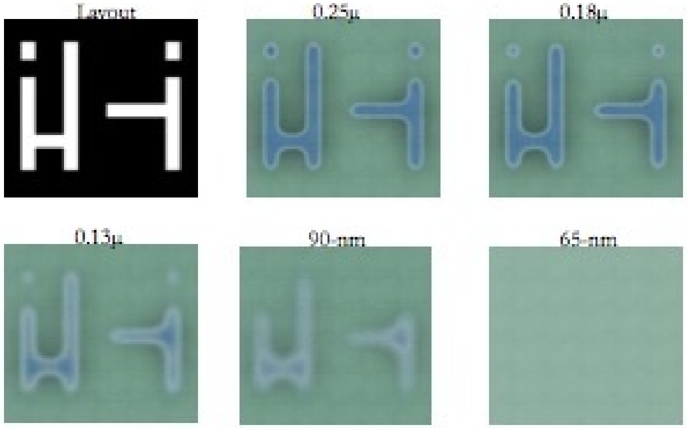
2000's: Stalled Speeds, Power^3

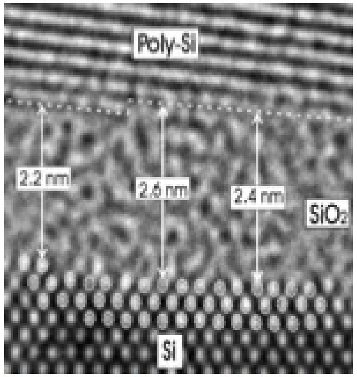
- 2010's: New Materials, Uncertainty or Variability



Source: Andrew Kahng

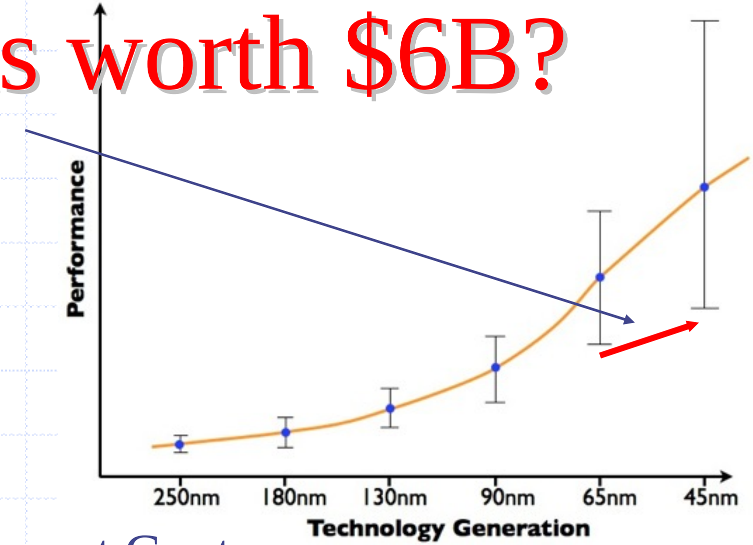
Variability Dominates The Mindset





Gate oxide variability

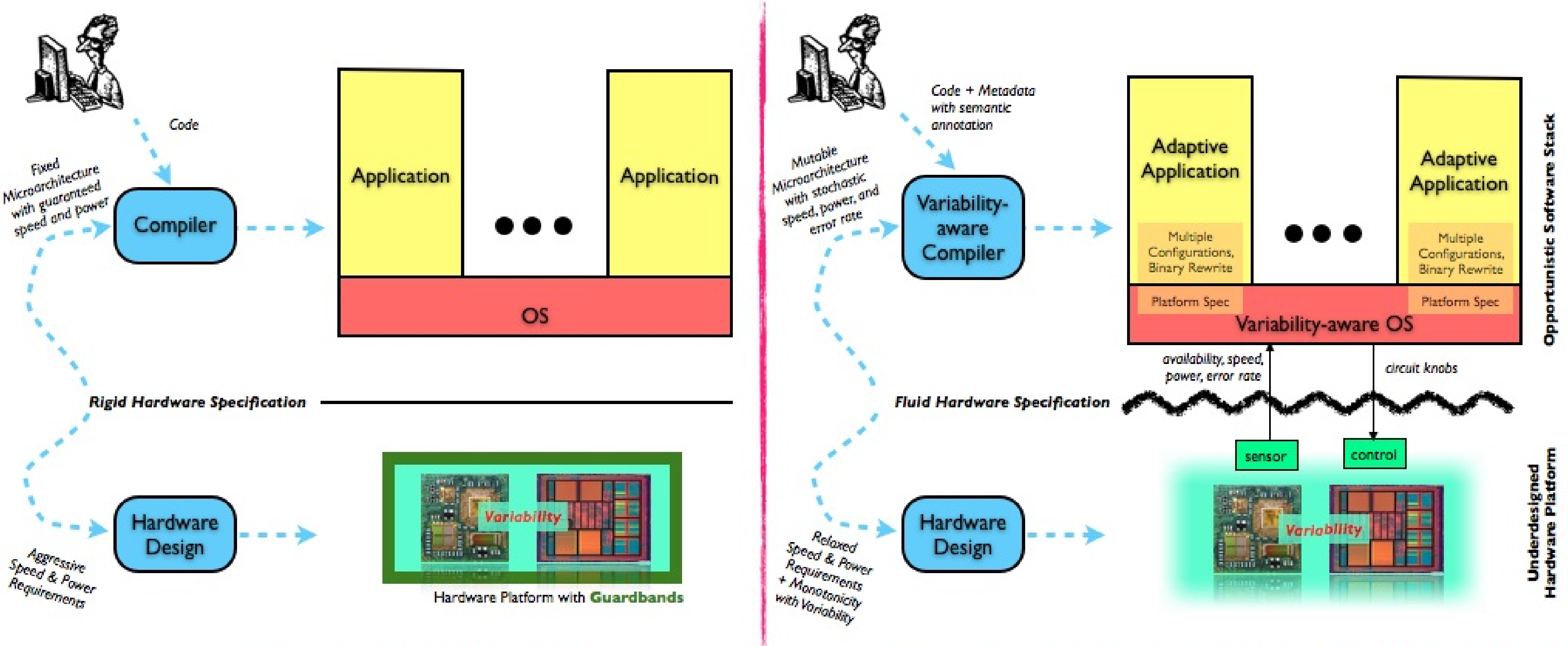
Is this worth $6B?



Source: Puneet Gupta

Long Term: Under-design Hardware!

- Reengineer SW stack to propagate variability and its response from OS to Applications.


Code
Fixed Microarchitecture with guaranteed speed and power
Compiler
Application
Application
OS
Rigid Hardware Specification
Hardware Design
Aggressive Speed & Power Requirements
Variability
Hardware Platform with Guardbands
Code + Metadata with semantic annotation
Mutable Microarchitecture with stochastic speed, power, and error rate
Variability-aware Compiler
Adaptive Application
Multiple Configurations, Binary Rewrite
Adaptive Application
Multiple Configurations, Binary Rewrite
Platform Spec
Platform Spec
Variability-aware OS
availability, speed, power, error rate
circuit knobs
Fluid Hardware Specification
sensor
control
Hardware Design
Relaxed Speed & Power Requirements + Monotonicity with Variability
Variability
Opportunistic Software Stack
Underdesigned Hardware Platform

Traditional Computing Machines

Underdesigned & Opportunistic (UnO) Computing Machines

Takeaways

1. Microelectronic realities are a driver of change in memory systems
   - Always driven by cost (scalability)

2. Programming has always been an unmet challenge, it only gets better

3. New generation of machines will see memory systems that are truly lightweight and flexible, courtesy meta-data handling methods from the Web.