

Homogeneous multiprocessing for the masses

Paul Stravers

Philips Research

Agenda

PART 1 Embedded multiprocessors

- Pollack's observation
- Power efficient processors
- Heterogeneous vs. homogeneous multiprocessors

PART 2 The shifting balance

- Technology scaling
- Efficiency gap is narrowing
- Platform success depends on flexibility
- Predictable system design

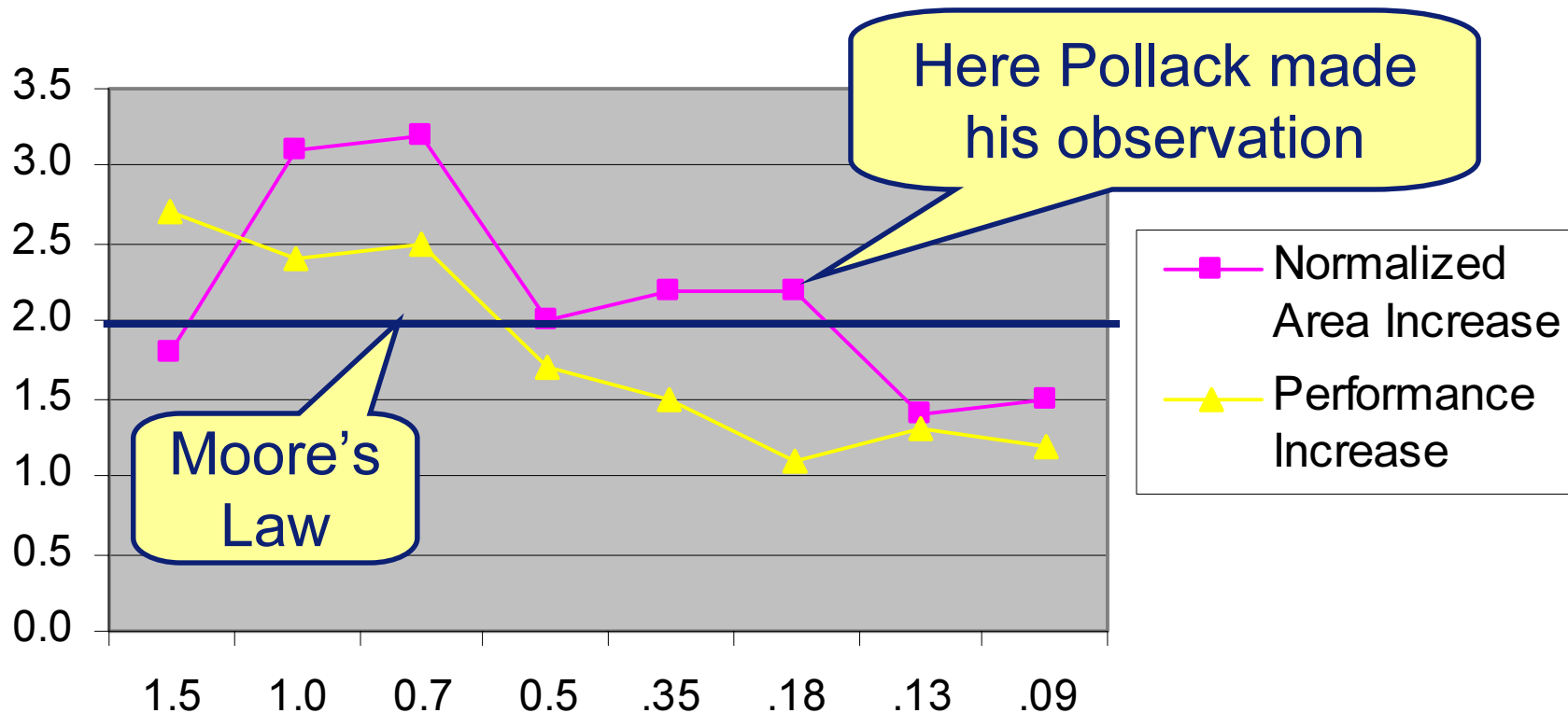
Pollack's observation

Pollack observed that the next processor architecture generation has:

Area: **2 ... 3x** equivalent gates

But only: **1.4 ...1.7x** more performance

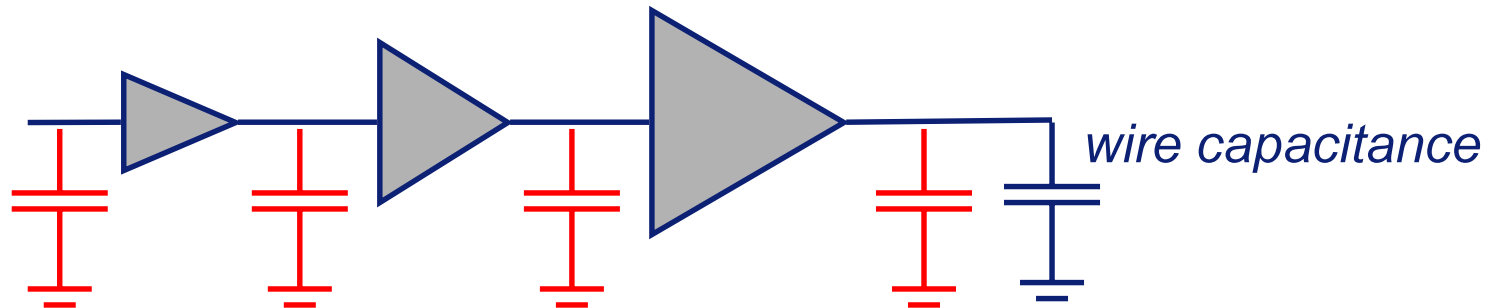
The situation is getting worse over time... *or is it?*



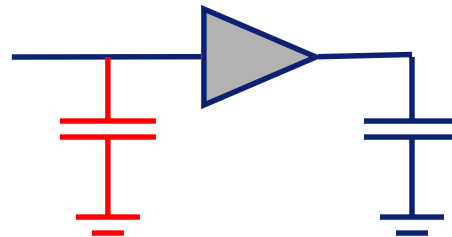
Power efficiency (1)

How to achieve high processor clock frequencies?

- Use strong buffers to quickly charge and discharge:



- But the buffers bring their own parasitic capacitances to the circuit, often much more than the wires!

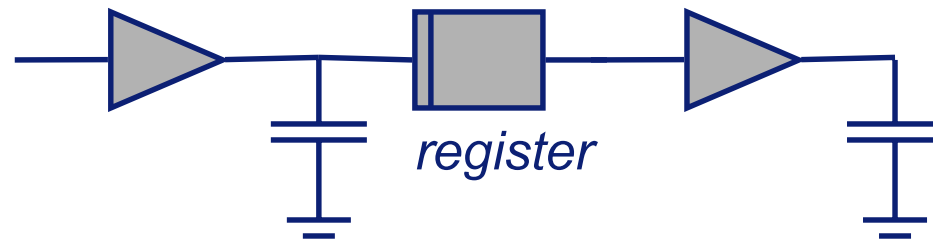


- Typical: **2x** faster clock \Rightarrow **4x** more Joules per clock !!!!

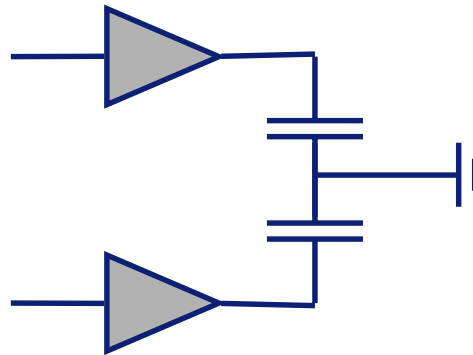
Power efficiency (2)

Solution is in parallelism: do multiple *independent* computations per clock cycle

- Pipelining (function parallelism):



- Instruction level parallelism (ILP) or data parallelism (SIMD):



Power efficiency (3)

Compare two industrial examples:

- Intel Pentium-4 (Northwood) in 0.13 micron technology
 - 3.0 GHz
 - 20 pipeline stages
 - Aggressive buffering to boost clock frequency
 - **13 nano Joule / instruction**
- Philips Trimedia “Lite” in 0.13 micron technology
 - 250 MHz
 - 8 pipeline stages
 - Relaxed buffering, focus on instruction parallelism
 - **0.2 nano Joule / instruction**
- Trimedia is doing **65x** better than Pentium

Power efficiency (4)

Who cares for power efficiency, anyway?

We do! ...Because

- Cost of electricity. The electricity bill of a modern PC is about 50 euro per year. Really!
- A chip package that can handle a hot processor chip costs up to 75 euro. Just the package!
- A fan (for air cooling) is expensive and noisy.
- The consumer appliance box gets ugly and expensive

Ultimately, heat dissipation limits what we can compute

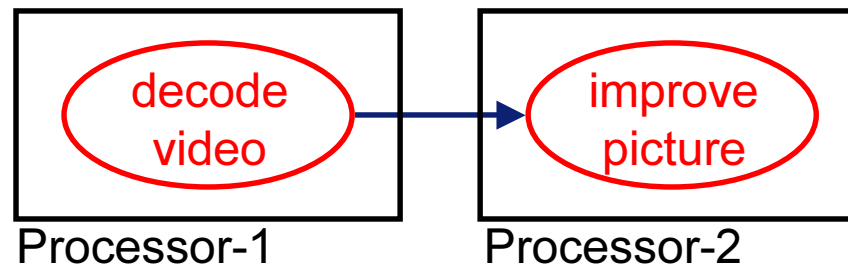
Parallelism

- Processor innovation has been driven for 40 years by **fine grain** parallelism:
 - **Pipelining**
 - **Instruction level parallelism (ILP)**
 - **Single instruction, multiple data (SIMD)**
- We are now confronted with diminishing returns (Pollack)
- The next big step is **course grain** parallelism
 - **Multiprocessing**
 - **Hyperthreading** (not discussed here)
- The key to parallelism is finding **independent** operations
- Course grain parallelism requires user awareness, i.e. the programmer must find the independent functions !!!

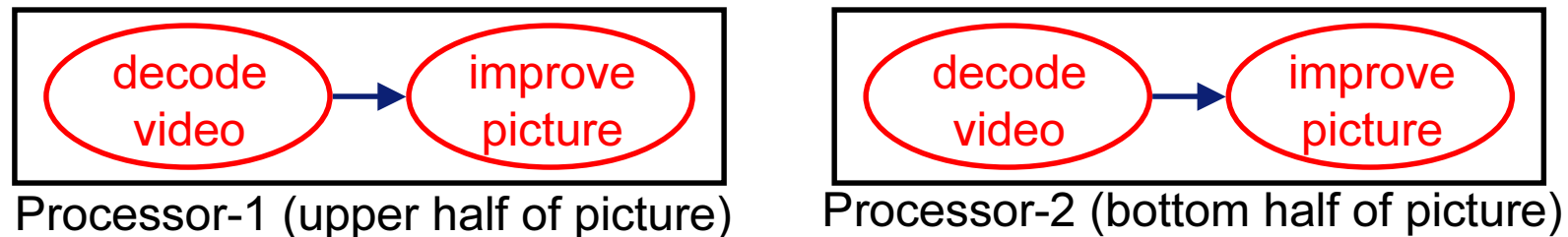
Multiprocessors (1)

Replace single big, hot processor with multiple small, efficient processors. Each computes part of the job:

Task parallelism:



Data parallelism:



Multiprocessors (2)

Two of the most important architecture characteristics are

- **Synchronization**

Tasks must wait on each other. For example:

- only read new data after it has actually been produced by the upstream task (task parallelism)
- only proceed with next picture when all tasks are done with current picture (data parallelism)

- **Communication**

Move data between tasks. Two basic models:

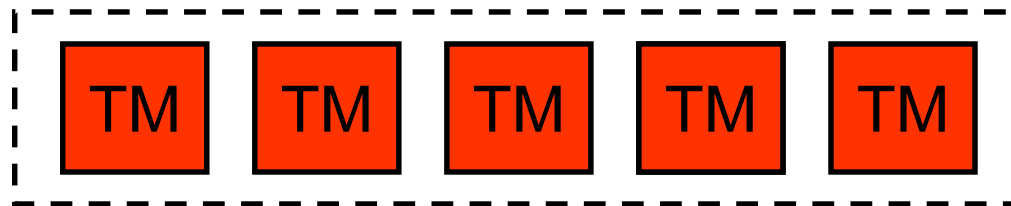
- Message passing: explicit action by sender & receiver
- Shared memory: implicit access by all tasks to all data

Heterogeneous vs. homogeneous multiprocessors (1)

Another important characteristic is the mix of processors.

Homogeneous:

Very flexible, moderate design effort, moderate comp. efficiency



Heterogeneous:

Less flexible, large design effort, high comp. efficiency



Heterogeneous vs. homogeneous multiprocessors (2)

The following characteristics play well together:

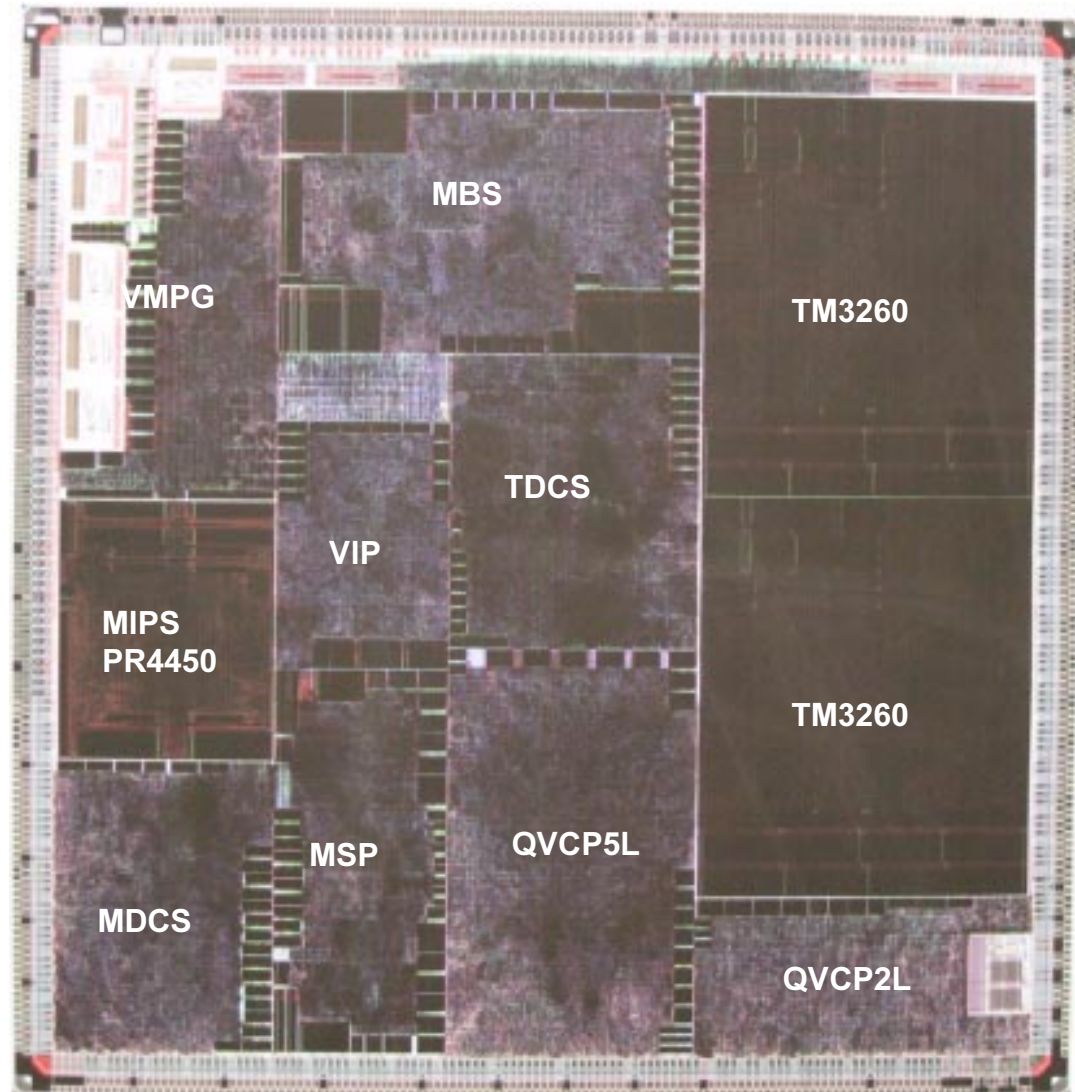
Homogeneous \Leftrightarrow **data parallelism** \Leftrightarrow **shared memory**
 \Leftrightarrow **dynamic task mapping**

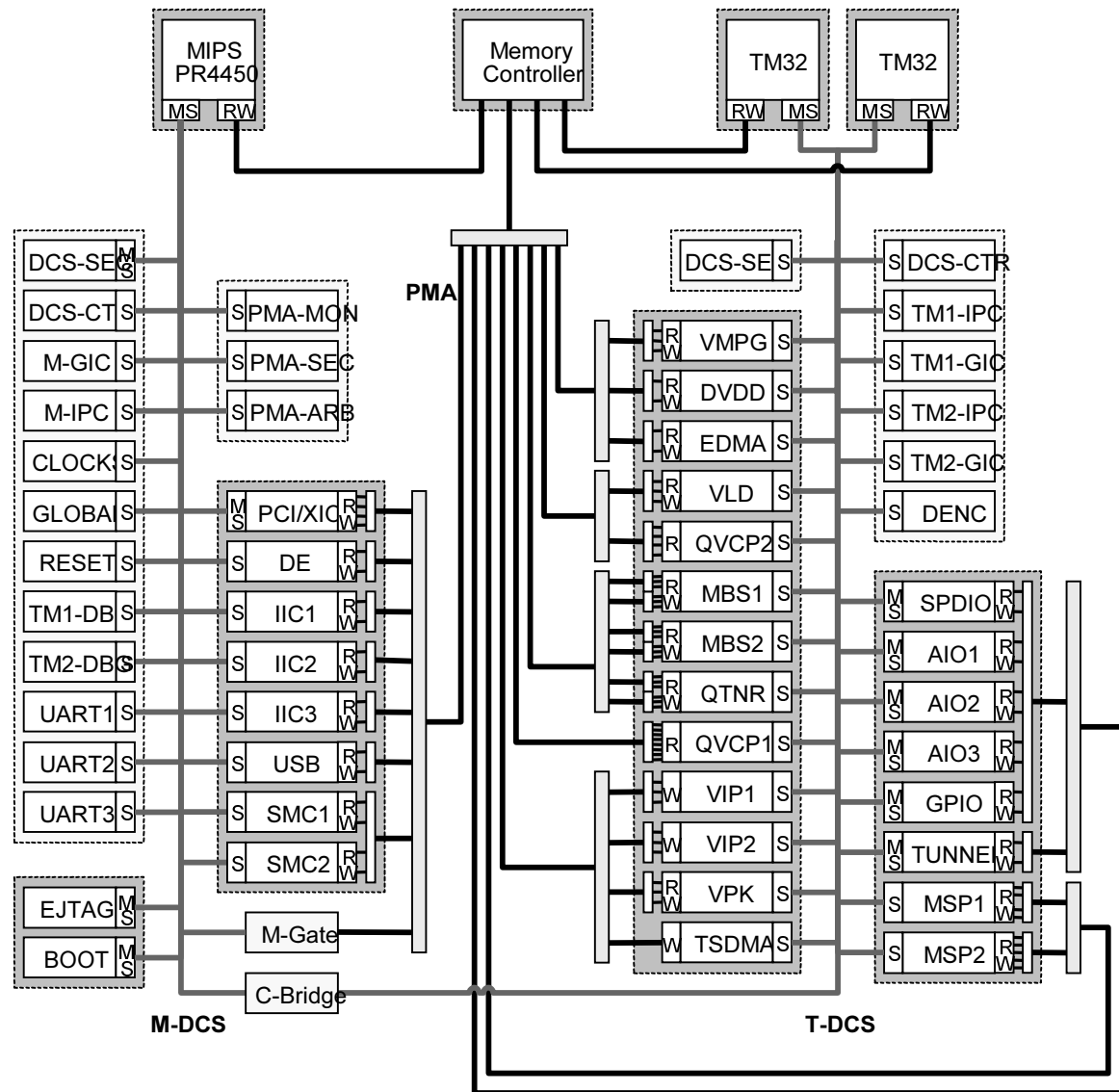
Heterogeneous \Leftrightarrow **task parallelism** \Leftrightarrow **message passing**
 \Leftrightarrow **static task mapping**

Historically,
embedded multiprocessors favor the heterogeneous model;
server multiprocessors favor the homogeneous model

Example: Viper2

- Heterogeneous
- Platform based
- >60 different cores
- Task parallelism
- Sync with interrupts
- Streaming communication
- Semi-static application graph
- 50 M transistors
- 120nm technology
- Powerful, efficient

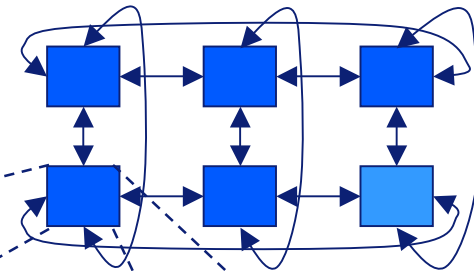




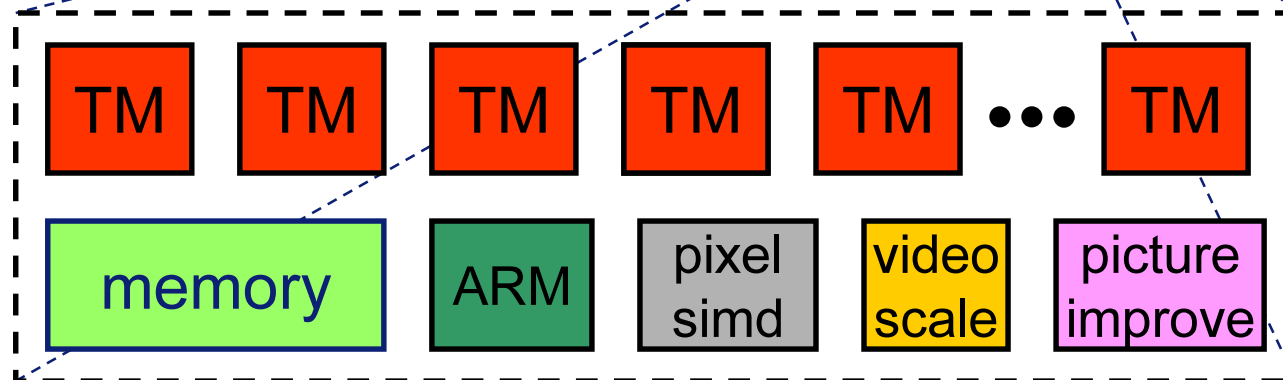
Example: Philips Wasabi

- Homogeneous multiprocessor for media applications
- First 65 nm silicon expected 1st half 2006
- Two-level communication hierarchy

- **Top:** scalable message passing network plus *tiles*

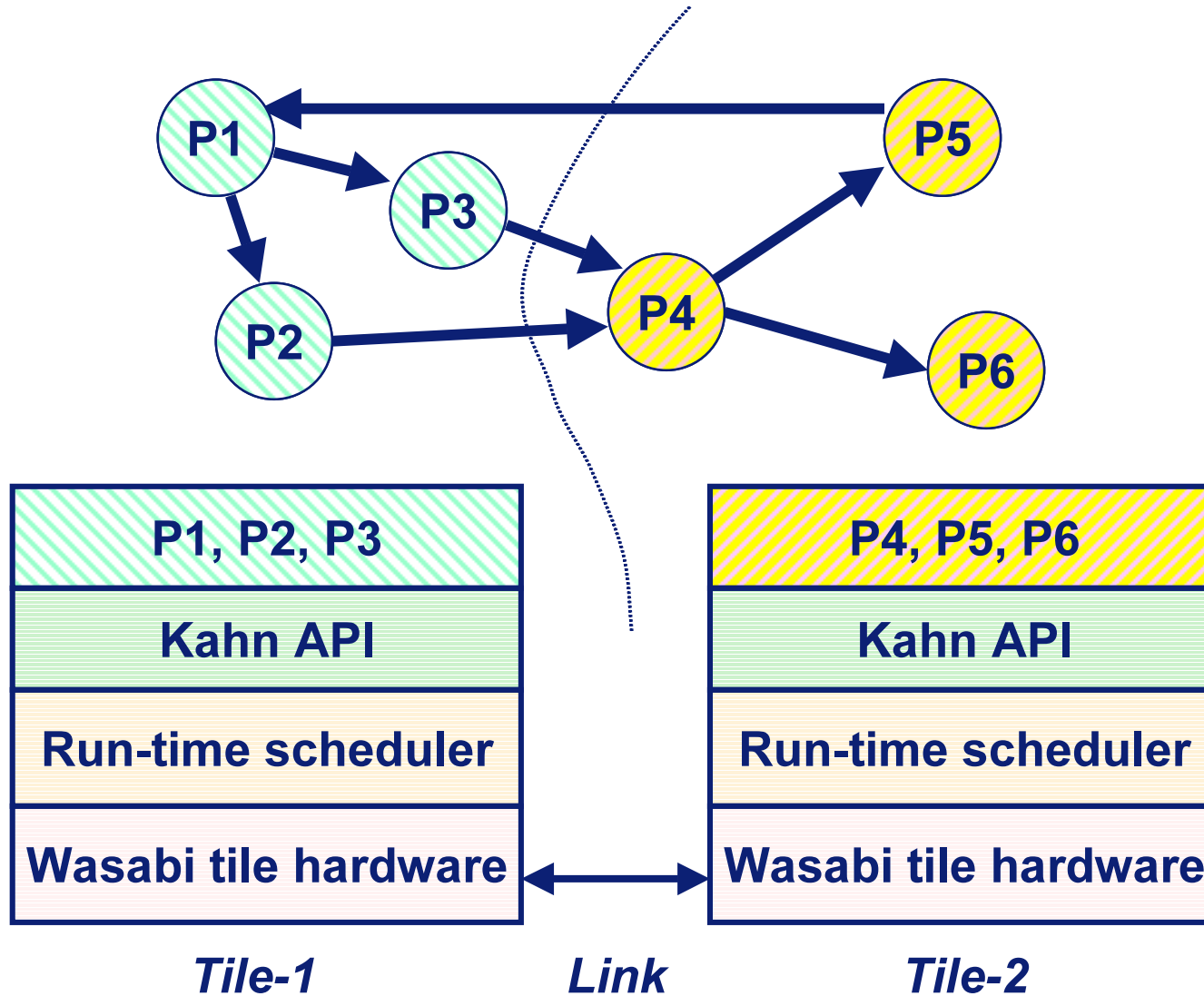


- **Tile:** shared memory plus processors, accelerators

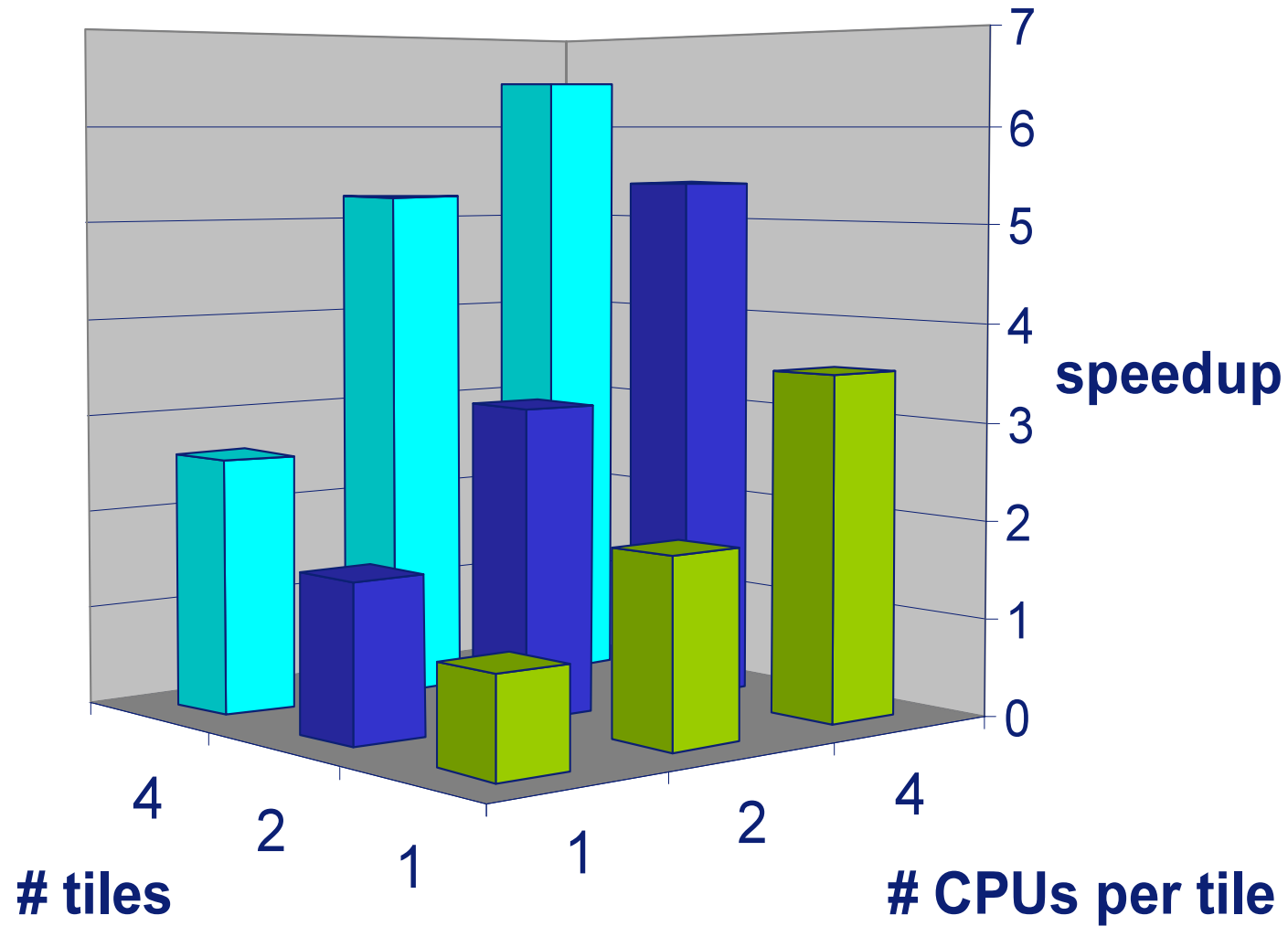


- Fully cache coherent to support data parallelism

Static task graph partitioning



Limited speedup with task parallel mpeg2 decoder



Data parallelism (1)

- We would like to do better than 5x speedup
- MPEG2: 1920x1088 picture contains at least 68 slices
- Each slice can be decoded *independently* from other slices
- Original code in function decode_picture():

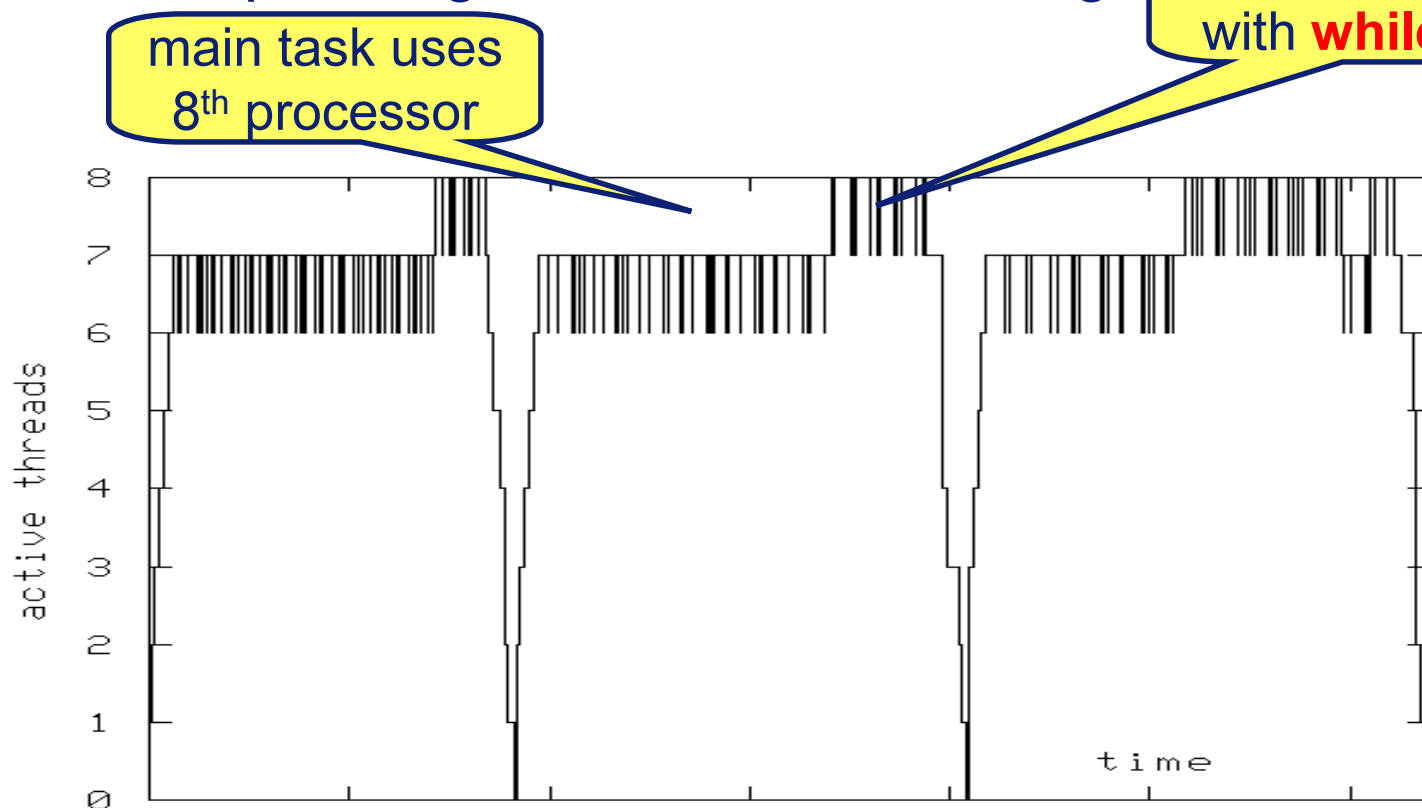
```
while (next_slice_start_code())  
    decode_slice();
```

- New code:

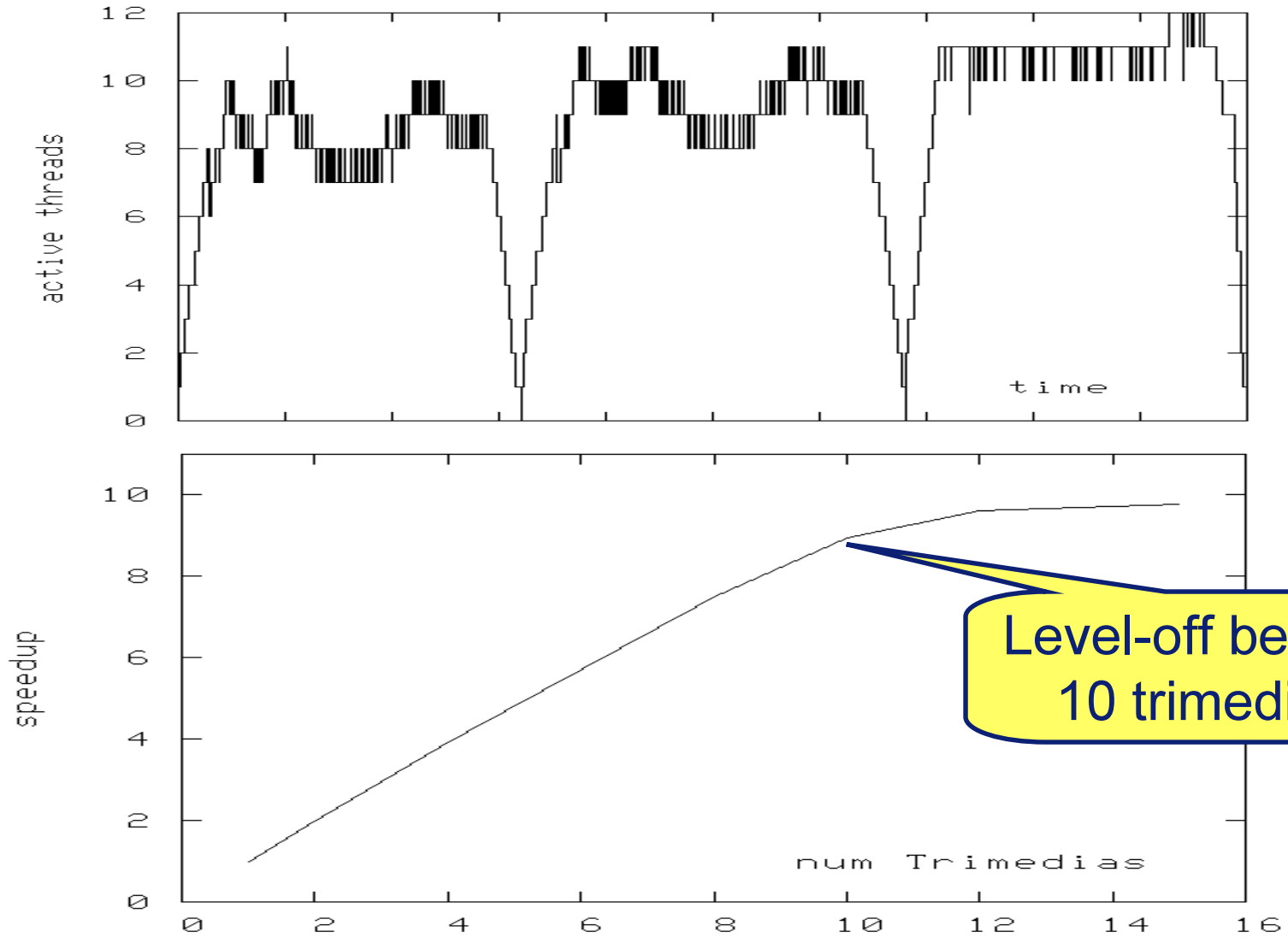
```
while (next_slice_start_code()) {  
    context[i] = copy_context();  
    new_task(decode_slice, &context[i]);  
    i = i + 1;  
}  
wait_for_all_unfinished_tasks();
```

Data parallelism (2)

- Create as many tasks as there are slices
- Run-time scheduler maps tasks to available processors
- Task surplus is good for load balancing!



Data parallelism (3)



Data parallelism (4)

Analysis:

The main task (executing the **while** loop) cannot find new slices fast enough to keep >10 trimedias busy

The underlying problem is a very inefficient implementation of **next_slice_start_code()**

With little effort we fixed the problem, resulting in near-linear speedup for 20 trimedias and up

Surprisingly, only 1 man-week was invested in modifying the original sequential code for data parallelism...

Multiprocessors summary

- Pollack's Rule calls for course grain parallelism
- Programmer must be aware of course grain parallelism!
- Multiprocessors can have ~2 orders better power efficiency
- Heat dissipation limits what we can compute
- Data parallelism scales better than task parallelism
 - Requires homogeneous multiprocessor
 - Easier to program when sequential program available (E.g. 1 man-week vs. 1 man-year for Kahn task graph)
- Heterogeneous architectures are even more power efficient but are less flexible and offer no data parallelism

Agenda

PART 1 Embedded multiprocessors

- Pollack's observation
- Power efficient processors
- Heterogeneous vs. homogeneous multiprocessors

PART 2 The shifting balance

- Technology scaling
- Efficiency gap is narrowing
- Platform success depends on flexibility
- Predictable system design

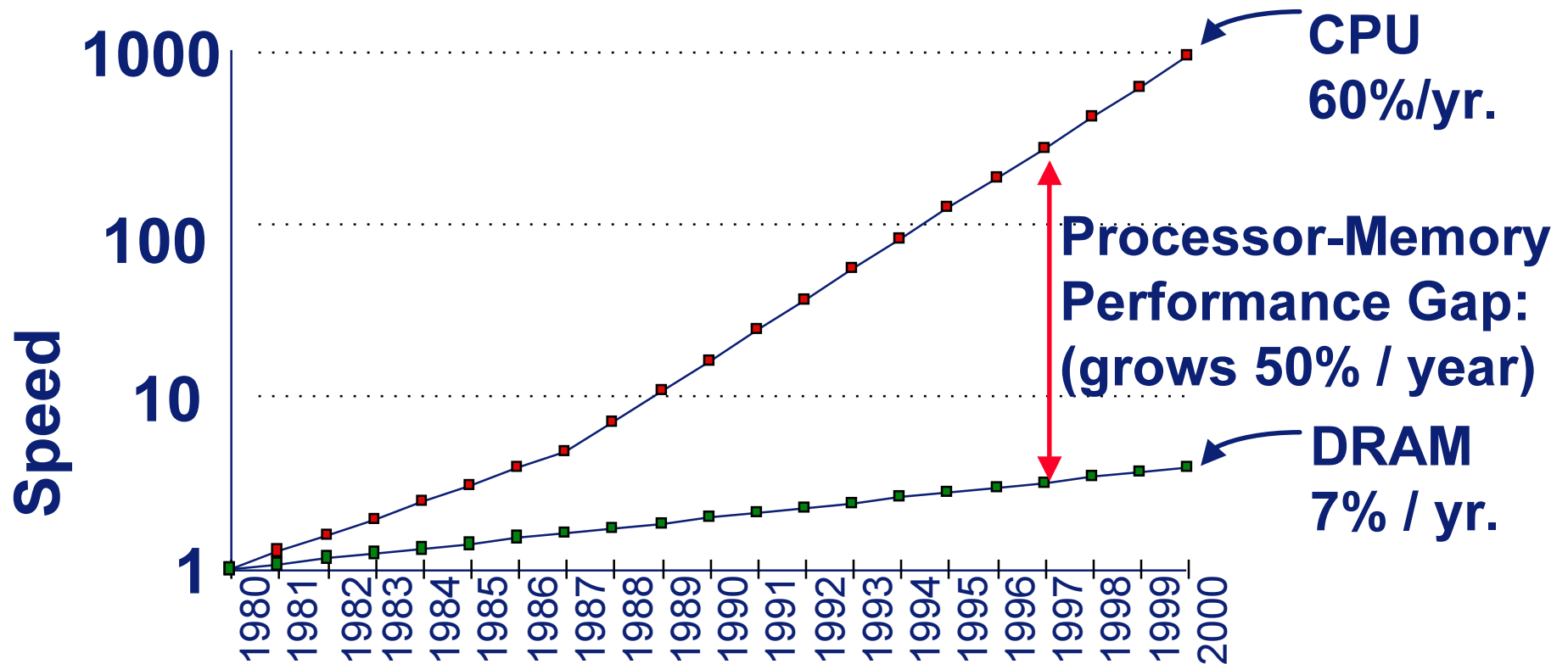
Effect of technology scaling on architecture

In the next few slides we show that technology scaling shifts the balance in favor of homogeneous, software-centered architectures:

- Memory increasingly dominates silicon cost (on-chip memory for bandwidth, Straverius' First Law for capacity)
 - Smaller weight to area penalty of homogeneous processor array
- *Platform* with large user community is needed to recover non-recurrent silicon costs, and reduce application development costs
 - Flexible architecture increases application range

Memory speed trend

[Hennessy & Patterson, 1996]



Implications (1)

$\text{CPI} \approx \text{cache_miss_rate} * \text{mem_latency}$

$\text{cache_miss_rate} \cong 1/\text{sqrt}(\text{cache_size})$

Keeping CPI constant:

$\Rightarrow \text{cache_size} \cong (\text{CPU_memory_gap})^2$

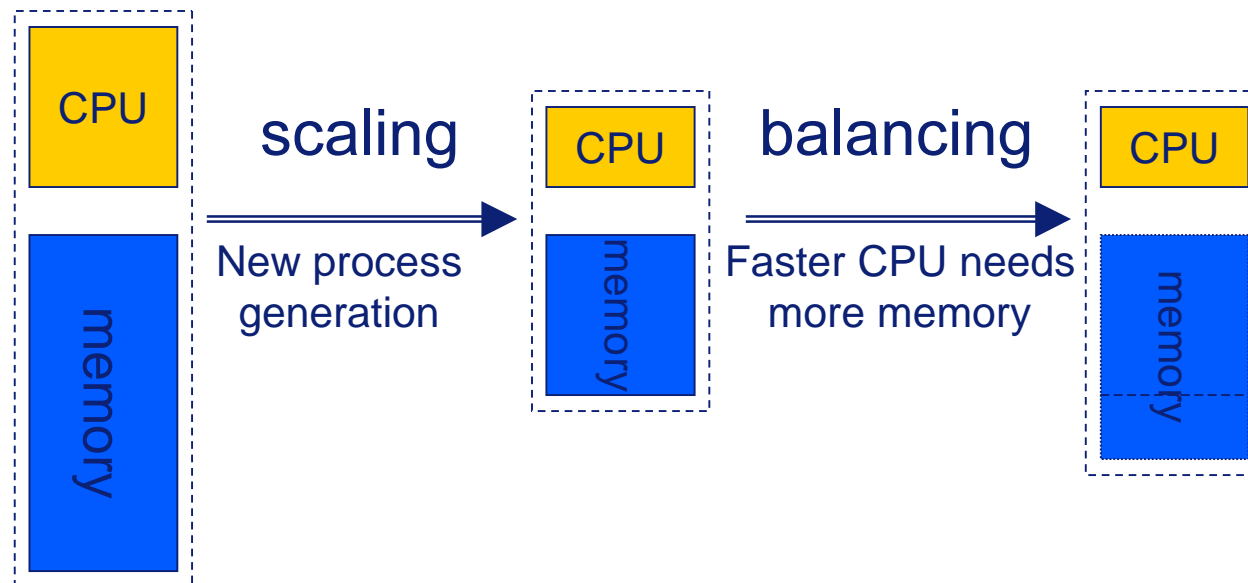
$\Rightarrow \text{cache_size} \cong (1.5)^2 / \text{yr} \approx \mathbf{2.2x / yr}$

Viper-1: **72 kB**

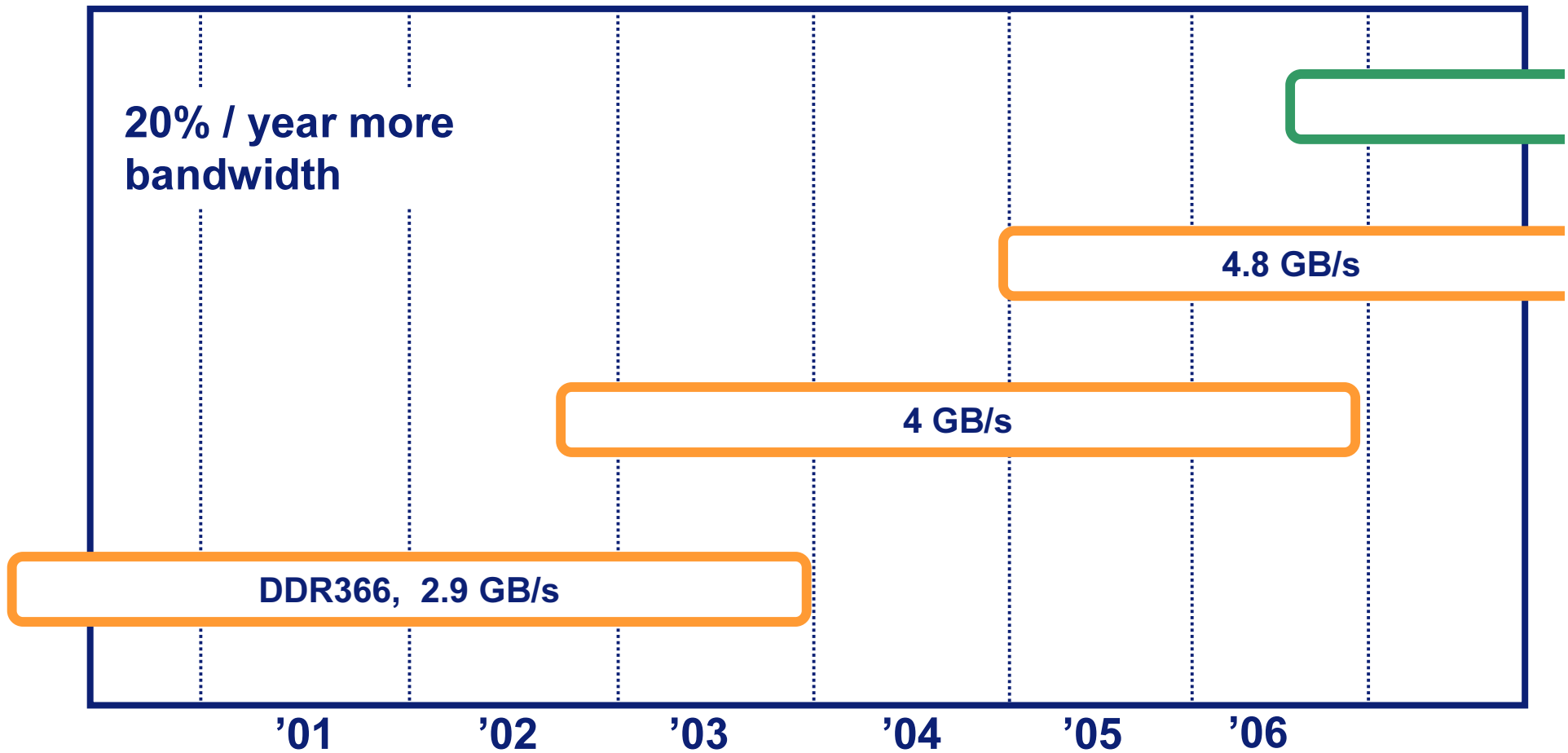
Viper-2: **192 kB**

Embedded DRAM speed

- Embedded DRAM frequency scales much better
- Straverius' Law takes over:
 - new technology node computes **S** times faster
 - need **S** times bigger cache, memory
 - results in **1.1x / yr** cache size increase



Memory bandwidth trend



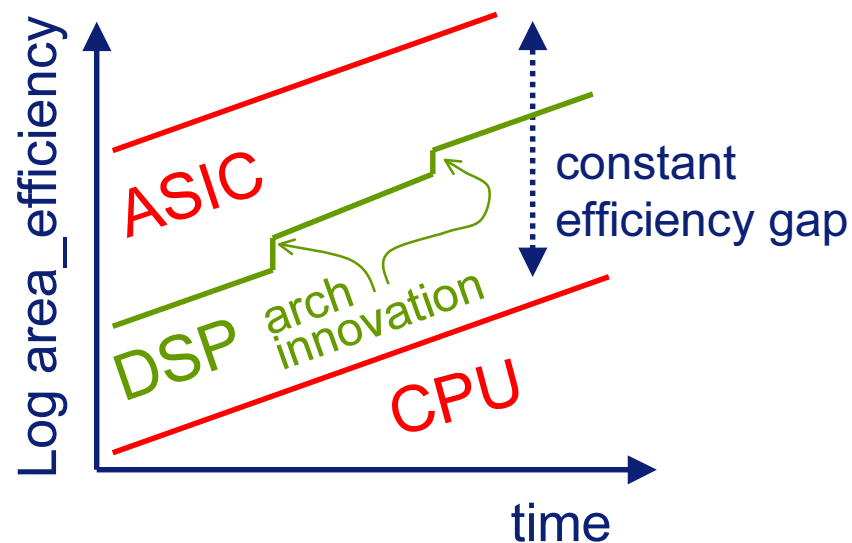
Memory bandwidth trend

- Compute speed, bandwidth: **60% / year**
- DDR, DDR2 bandwidth: **20% / year**

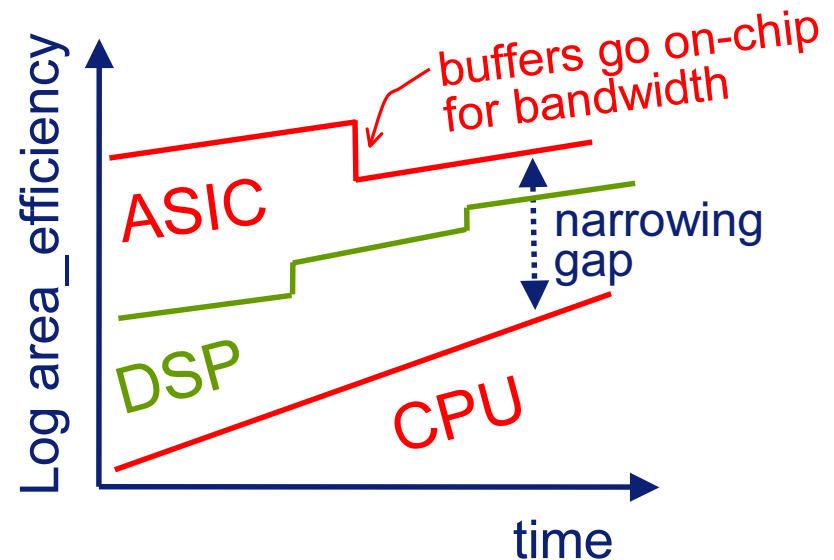
Bandwidth mismatch increasing 35% / yr

Efficiency gap is narrowing

- Memory footprint is predominantly determined by the *application*, much less by the architecture
- Technology scaling gradually replaces the intrinsic computation component with memory



intrinsic efficiency



system efficiency

What is a platform (1)

A platform is a collection of assets that are shared by a set of products. These assets can be divided into four categories:

- Components
- Processes
- Knowledge
- People and relationships

A successful platform needs all four ingredients!

Source: Robertson and Ulrich

What is a platform (2)

- The platform approach contrasts with *product-specific* development:
 - define and study the product requirements
 - then develop optimal technology for it
- Such dedicated products are potentially of higher quality than their platform counterpart
- But in practice one-time development endeavors often result in product flaws due to lack of experience
- Platforms, in contrast, can be improved thru feedback from earlier products

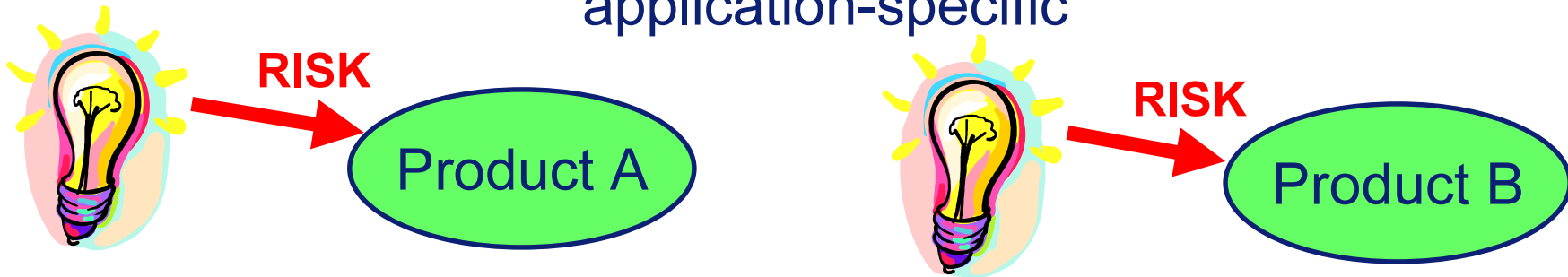
Platform benefits

- Greater ability to tailor products
- Reduced development cost and time
- Reduced manufacturing cost
- Reduced production investment
- Reduced system complexity
- Lower risk
- Improved service

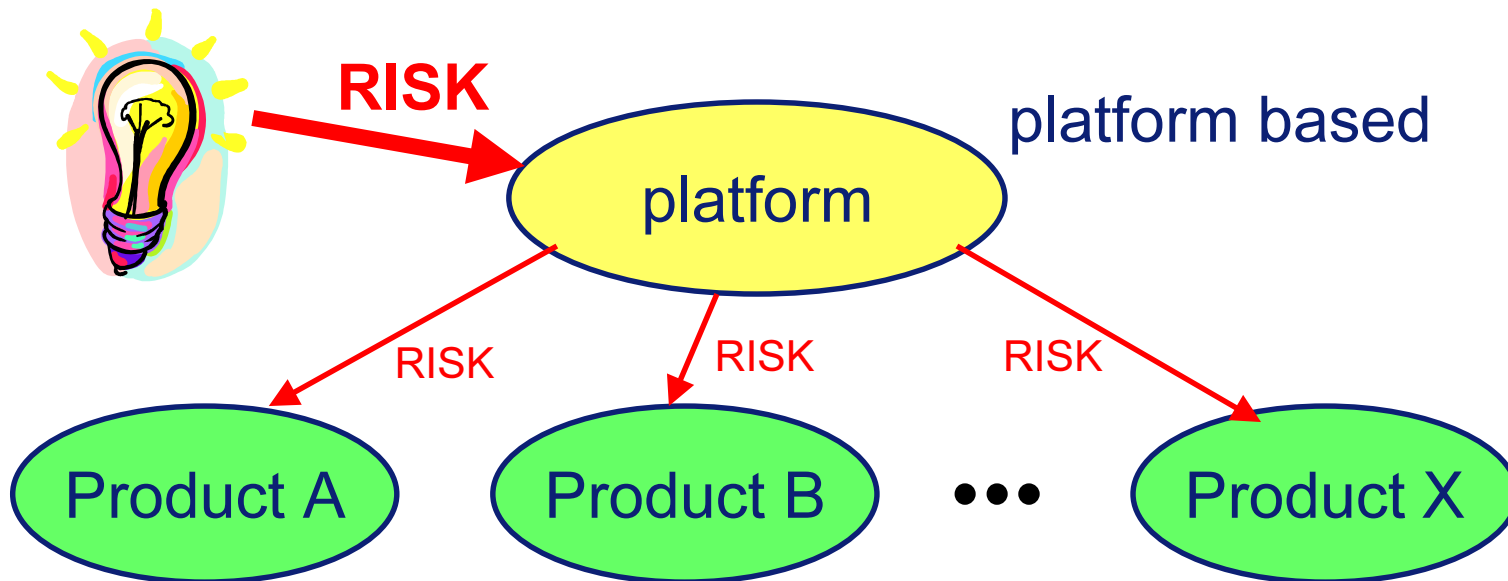
Source: Robertson and Ulrich

Platform risks (1)

application-specific



platform based

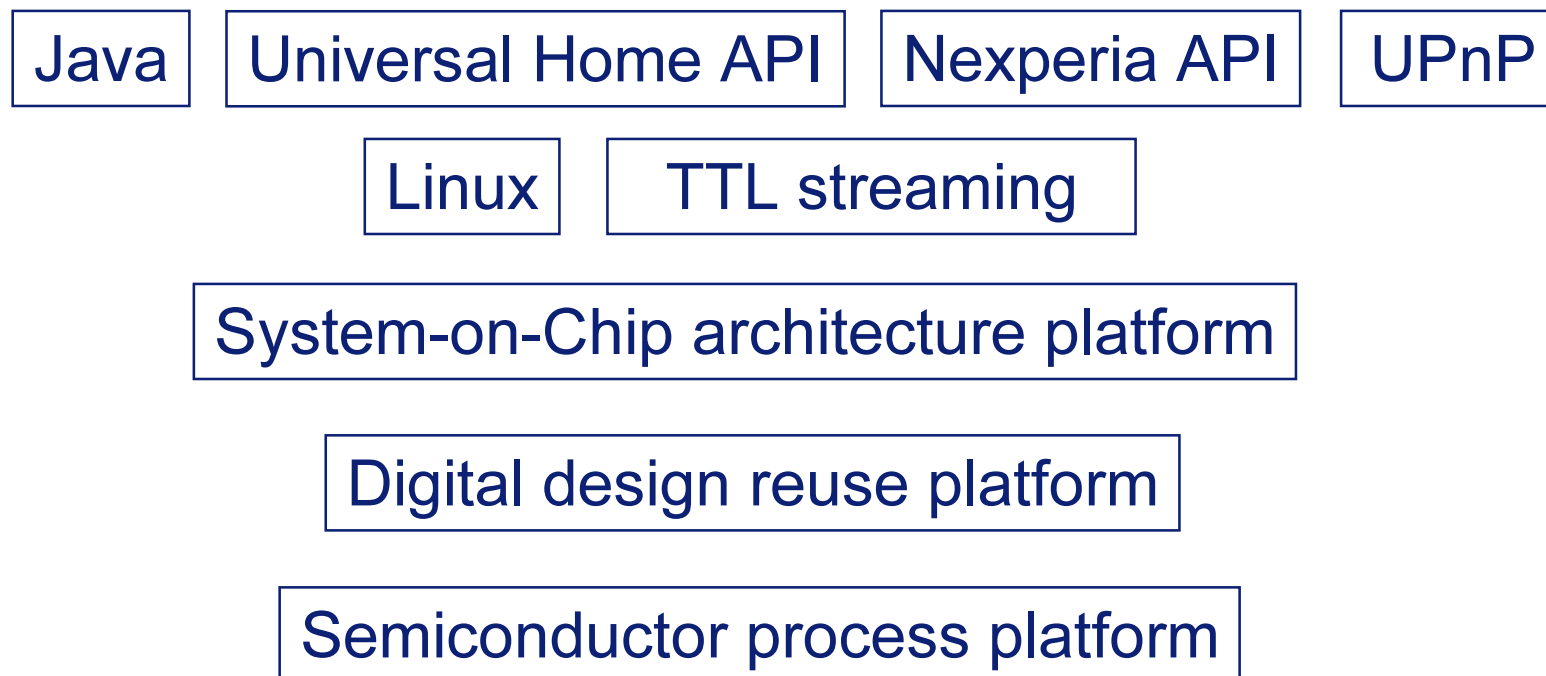


Platform risks (2)

- Platform benefits only materialize if all ingredients are present
 - components, processes, knowledge, community
- A lot of effort (=money) goes into platform building
 - components, processes, knowledge
- Pay-back only occurs when enough products are derived
- This requires a large enough developer community
- Typically this means that company-internal platforms do not provide the expected benefits

⇒ Must share the platform with customers, even competitors!

A stack of platforms in consumer media products



Putting it together

- Homogeneous systems are inherently more flexible than heterogeneous systems
 - address broader application range
 - attract a larger user community
- Strong need for successful SoC *platform*
 - Non-recurrent costs keep rising
 - Product innovation rate keeps rising
- Efficiency gap is narrowing

Future platforms must focus on homogeneous architecture

Predictable system design

- Heterogeneous systems have more predictable behavior than homogeneous systems
 - Static task graph mapping
 - Dedicated streaming hardware functions
 - In general: less resource sharing
- Homogeneous embedded systems call for increased focus on shared resource virtualisation
 - Shared L2 caches: footprint and bandwidth
 - Bursty, latency critical memory traffic

Concluding remarks and hints for future research

- Homogeneous multiprocessors provide
 - A strong base for a successful media *platform*
 - Increasingly efficient processing power
 - Performance scaling, from low cost to high performance
 - Technology scaling (tiling), complement Moore's Law
 - High degree of software reuse (stable SoC architecture)
 - Very high silicon yield (redundant SoC architecture)
- More research needed on composable resource sharing
- More research needed on structured approach to shared memory media processing

