Programming Heterogeneous Many-core platforms in Nanometer Technology: *the P2012 experience*

Luca Benini

STMicroelectronics & Università di Bologna

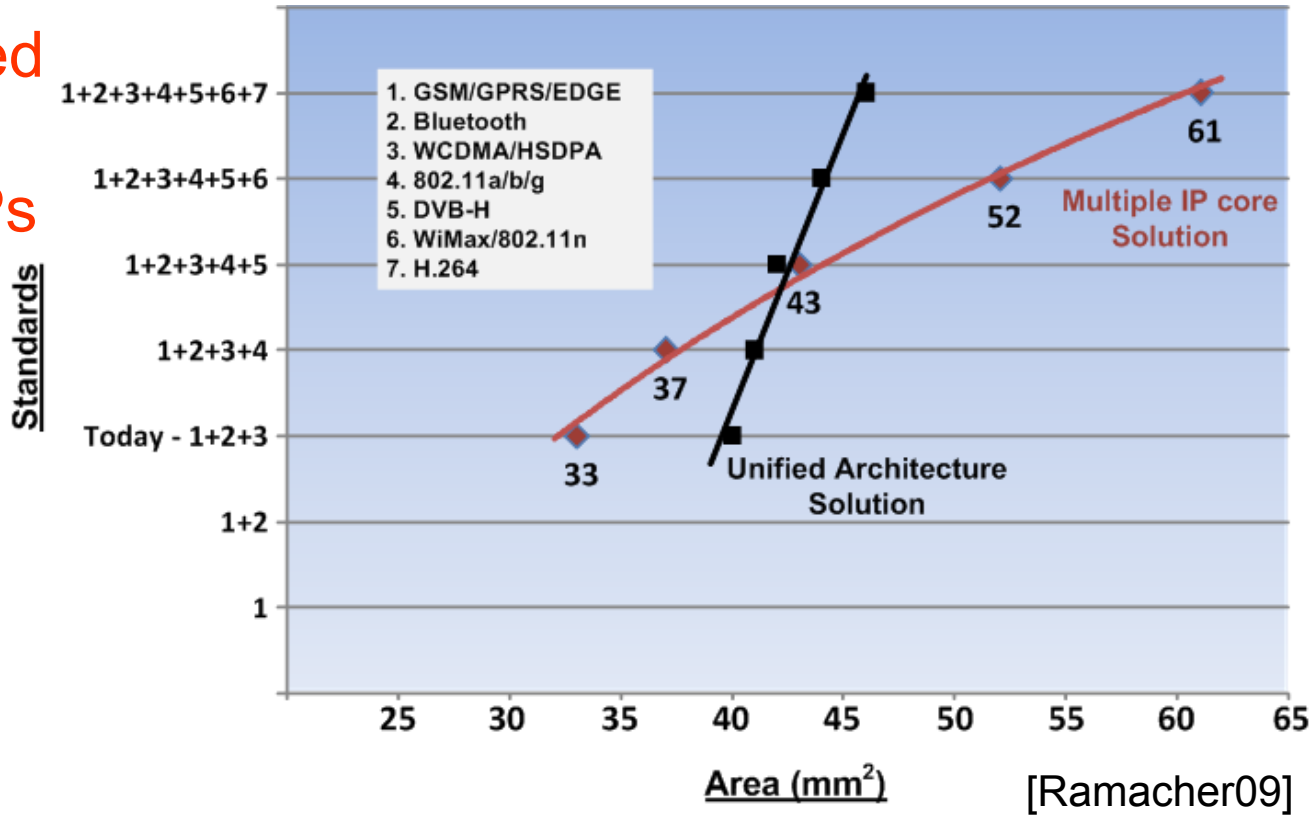
ARTIST DESIGN Summer school
Autrans, France, Sept 6th 2010

SoCs for Digital Convergence

Tight power and silicon area budgets

Flexibility is mandatory

Programmable Unified Architectures vs. multiple hardwired IPs



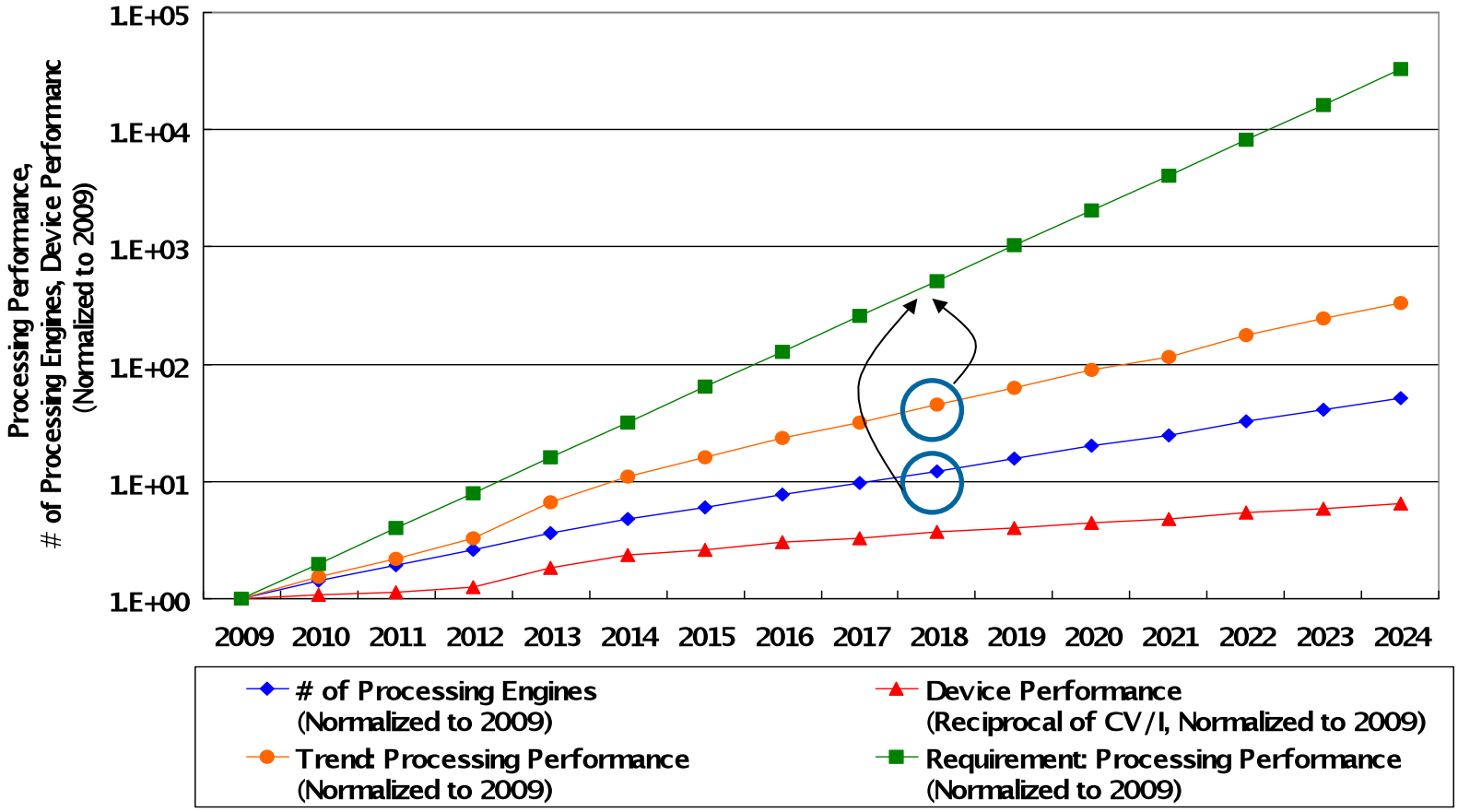
[Ramacher09]

Key advantages

- Lower Cost
- Faster Time to Market
- Support for Multiple Applications (Current and Future)
- Bug Fixes After Manufacturing

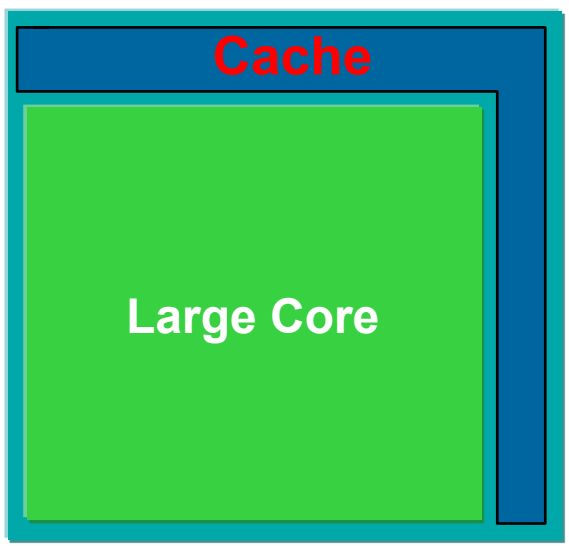
The multy-core Moore's Law

Multi/Many cores to bridge the gap between individual processing performance and system processing performance requirement

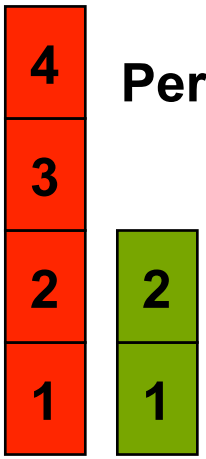


SOC Consumer Portable Processing Performance Trends. Source: ITRS

Multi-core and Silicon Efficiency



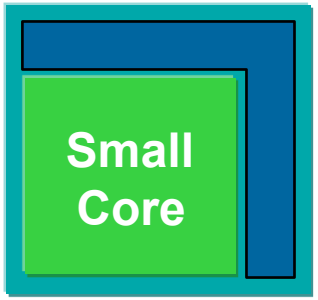
Power

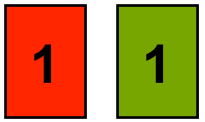


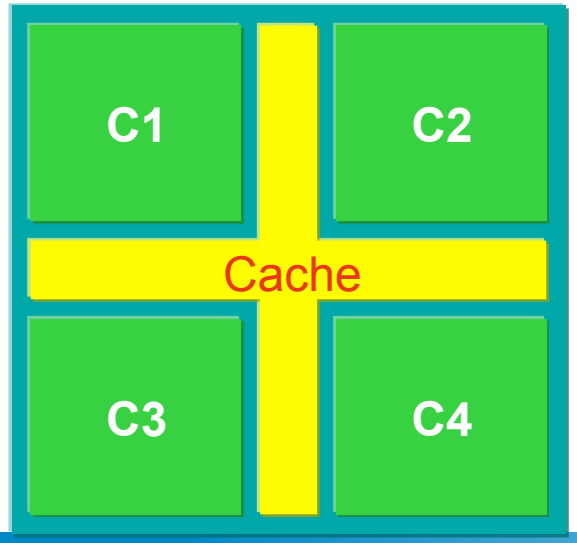
Performance

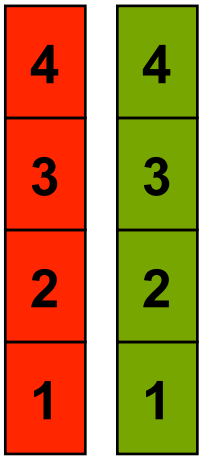
Power = 1/4

Performance = 1/2





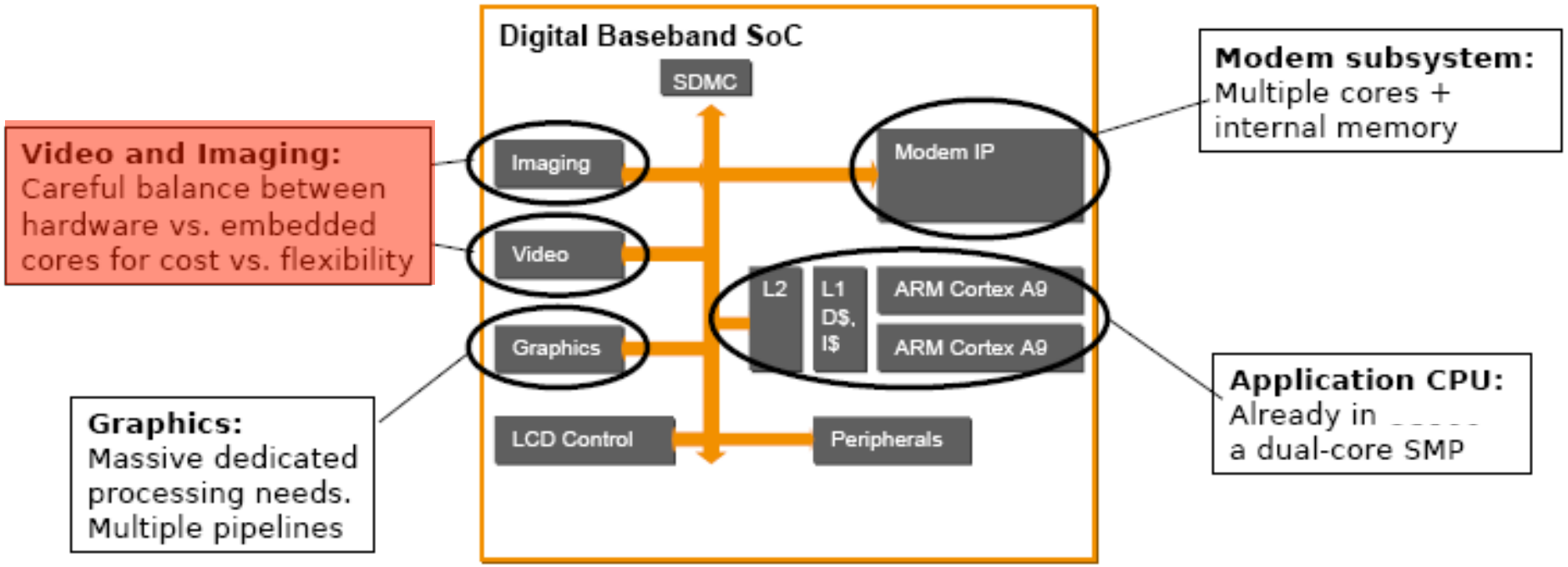




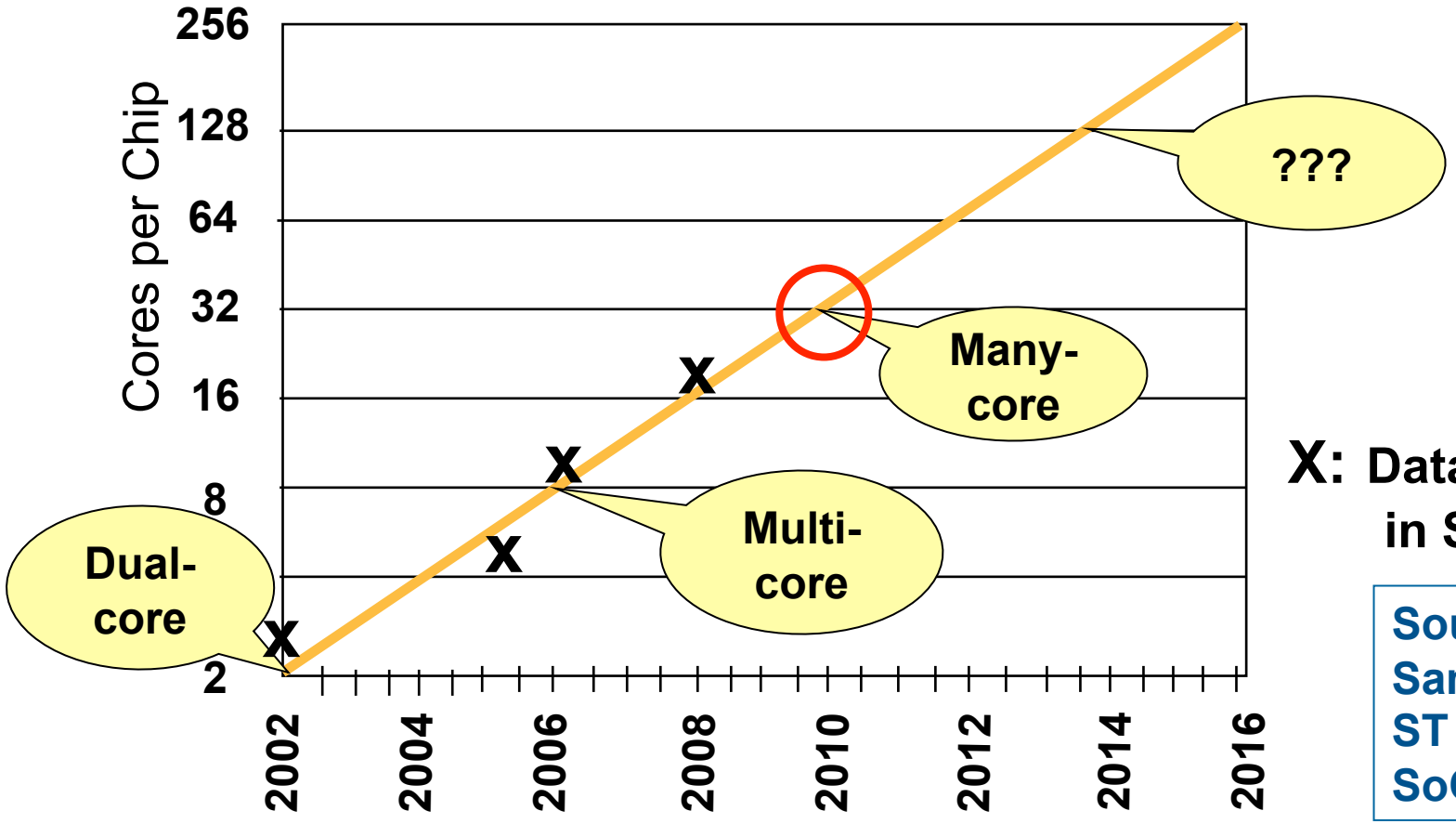
Multi-Core: Better power & computational density

Integrated SoC (AP+BB)

- High-speed SMP for “almost sequential” GP
- “Processor arrays” for domain-specific throughput computing (100x GOPS/W) **Our focus today**



Core's Law in ST



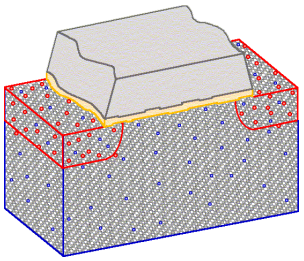
- Embedded cores in SoC products 2x every ~2 years
- Total SoC area very stable across tech nodes

Challenges

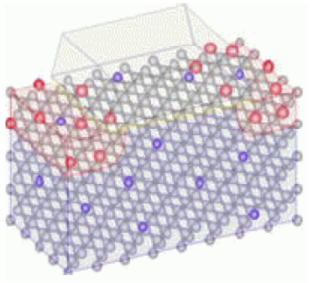
Yield and variability → Modular Fabric

Next generation process variations

- **The next generation of MOSFETs are atomic-scale devices**
- **Intra-die variations becomes significant**
  - Random Dopant Fluctuation
  - Across Chip Length Variation
  - RC variations
- **Over-design leading to higher power consumption and larger chip**
- **Yield likely to decrease for complex non regular design**

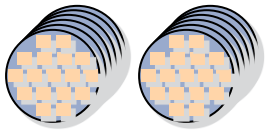


22 nm MOSFET

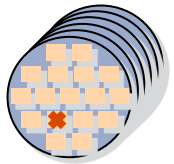


4 nm MOSFET

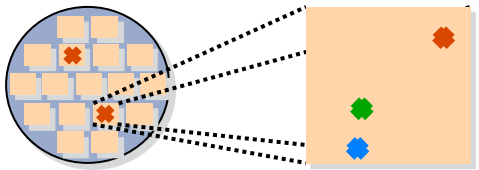
Lot-to-lot



Wafer-to-wafer

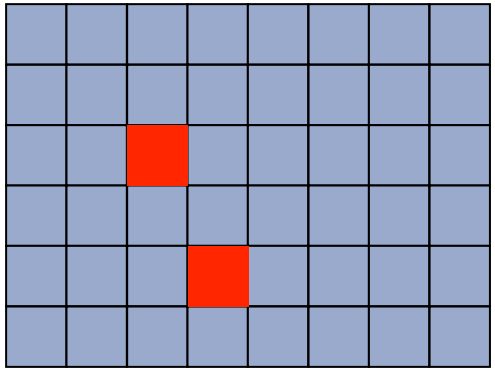


Die-to-die **Intra-Die**

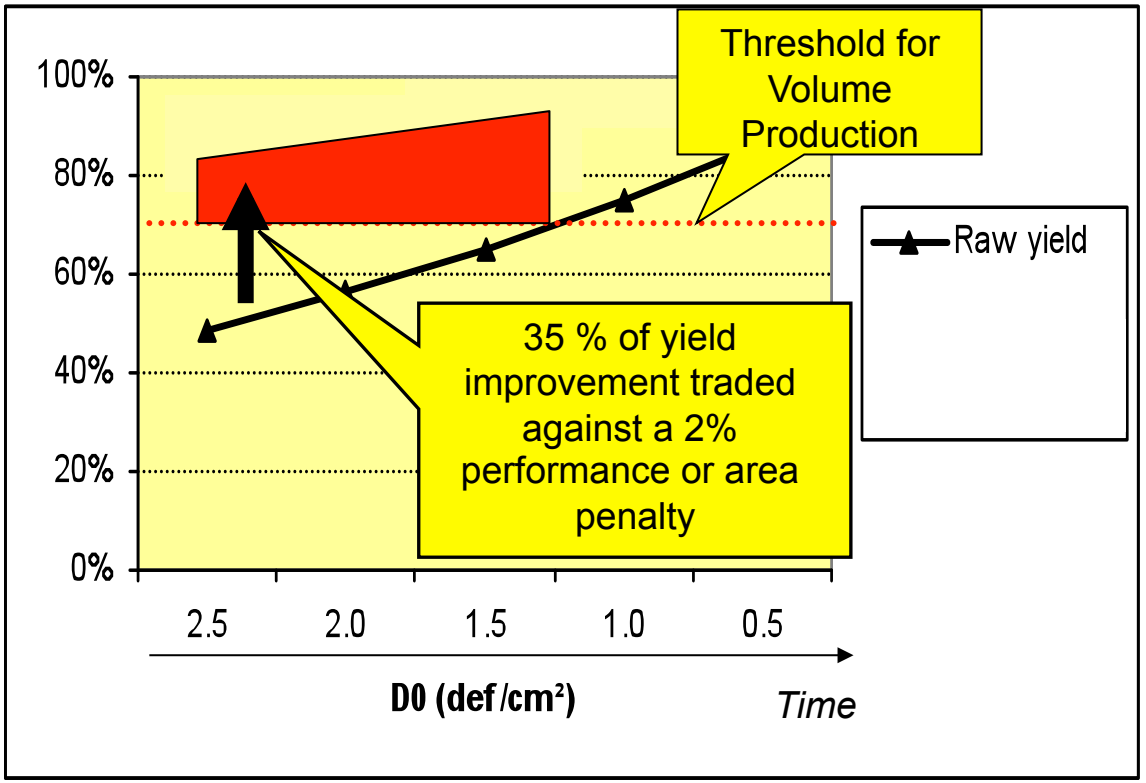


Manufacturing Yield

System → Computing islands



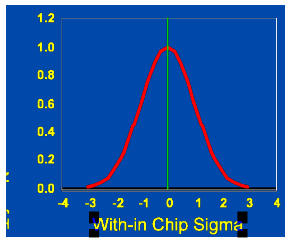
Computing Islands are decoupled. Non-functional ones can be removed at test time or even dynamically. Spare parts can be provisioned at design time.



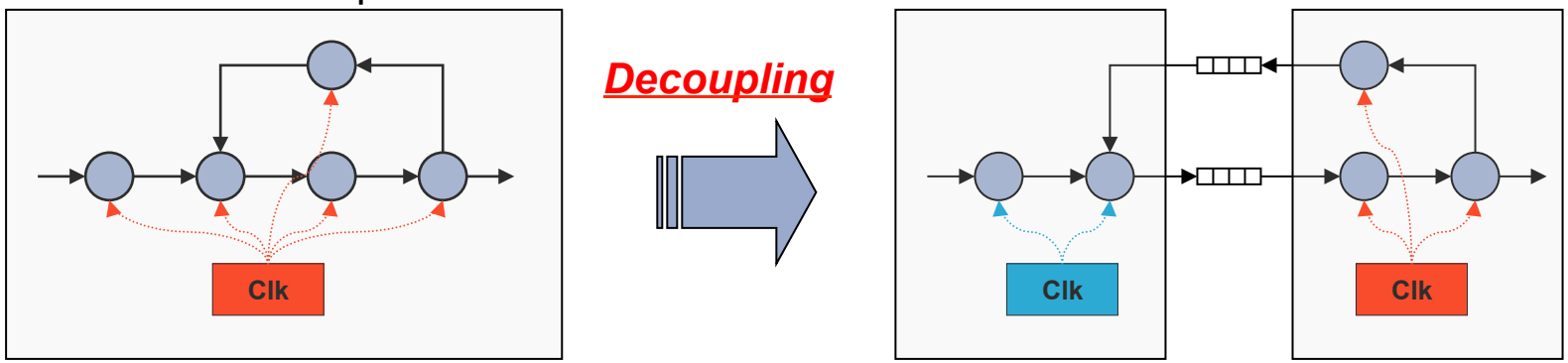
Lifetime increase of a process node
Earlier process node migration possible

Parametric Yield (variability)

Root cause: Variability



Clock and voltage can be adapted for each domain correcting local variations of the process

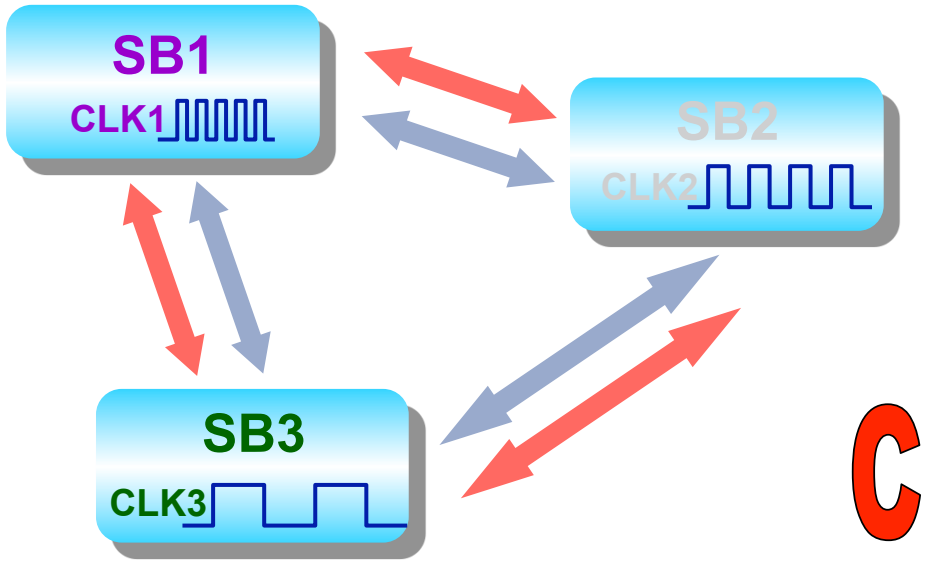


[Garg DATE07]

Yield gain at same throughput with limited power impact + Wearout Mgt

Decoupling →GALS

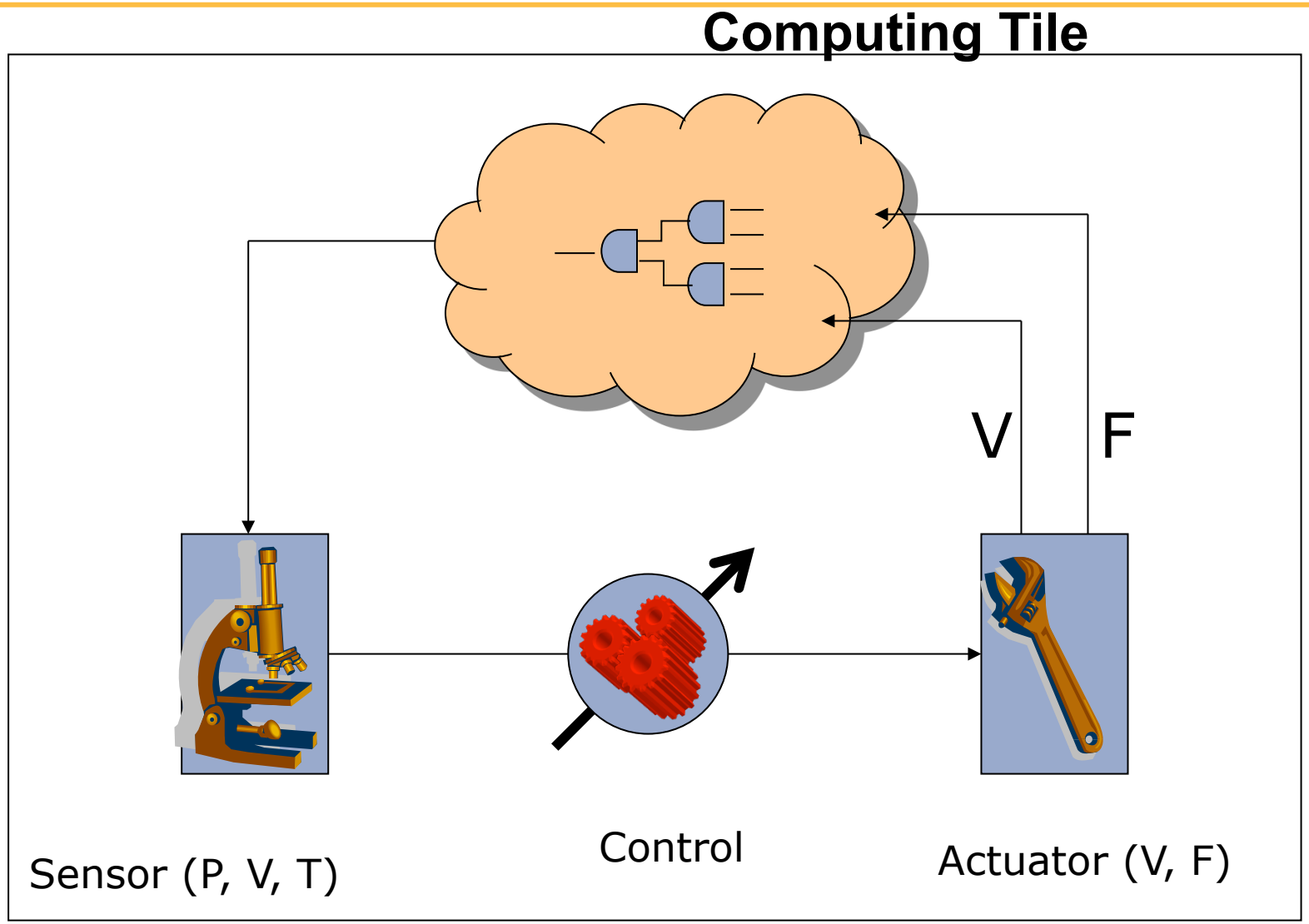
- Several synchronous clock domains (Synchronous Blocks) communicate through asynchronous channels
- Each SB can be clocked independently for local needs



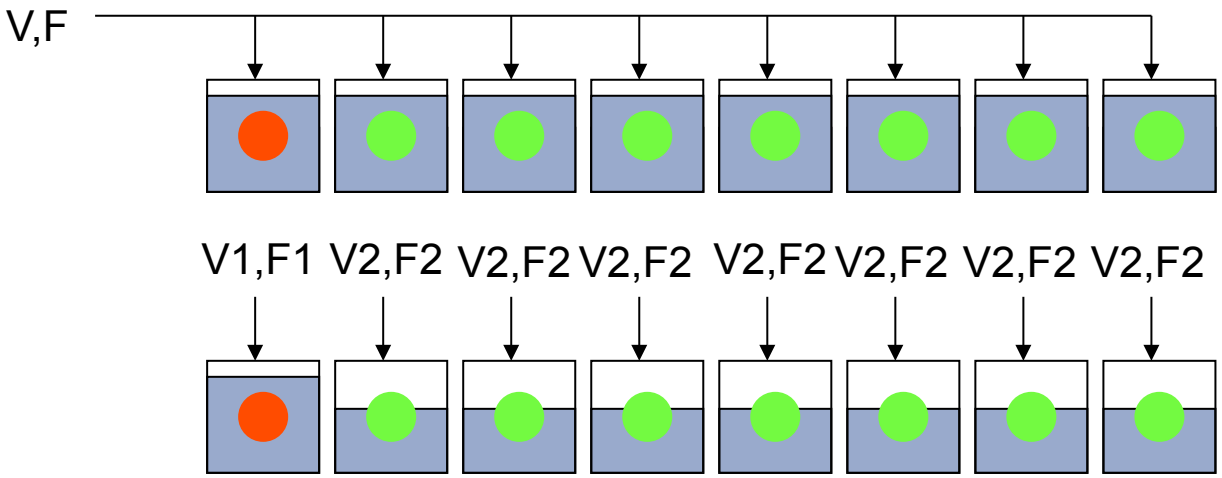
- Pros
  - ✓ Reduced average frequency
  - ✓ Reduced global power
  - ✓ Globally skew tolerant
  - ✓ No global clock in GALS: easier local clock design
- Cons
  - × Global communication overhead from handshake protocol
  - × Total area, average current and power penalty from protocol
  - × Local clock overhead

Can be dealt with

Closed Loop Control

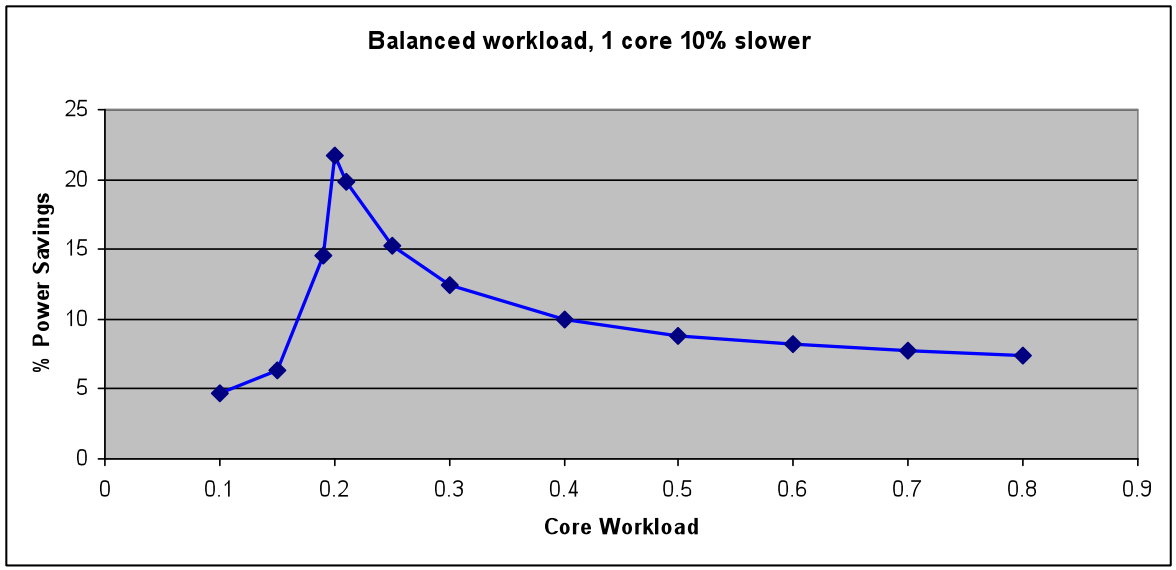


Effect of local adaptation on power



Coupled

Decoupled

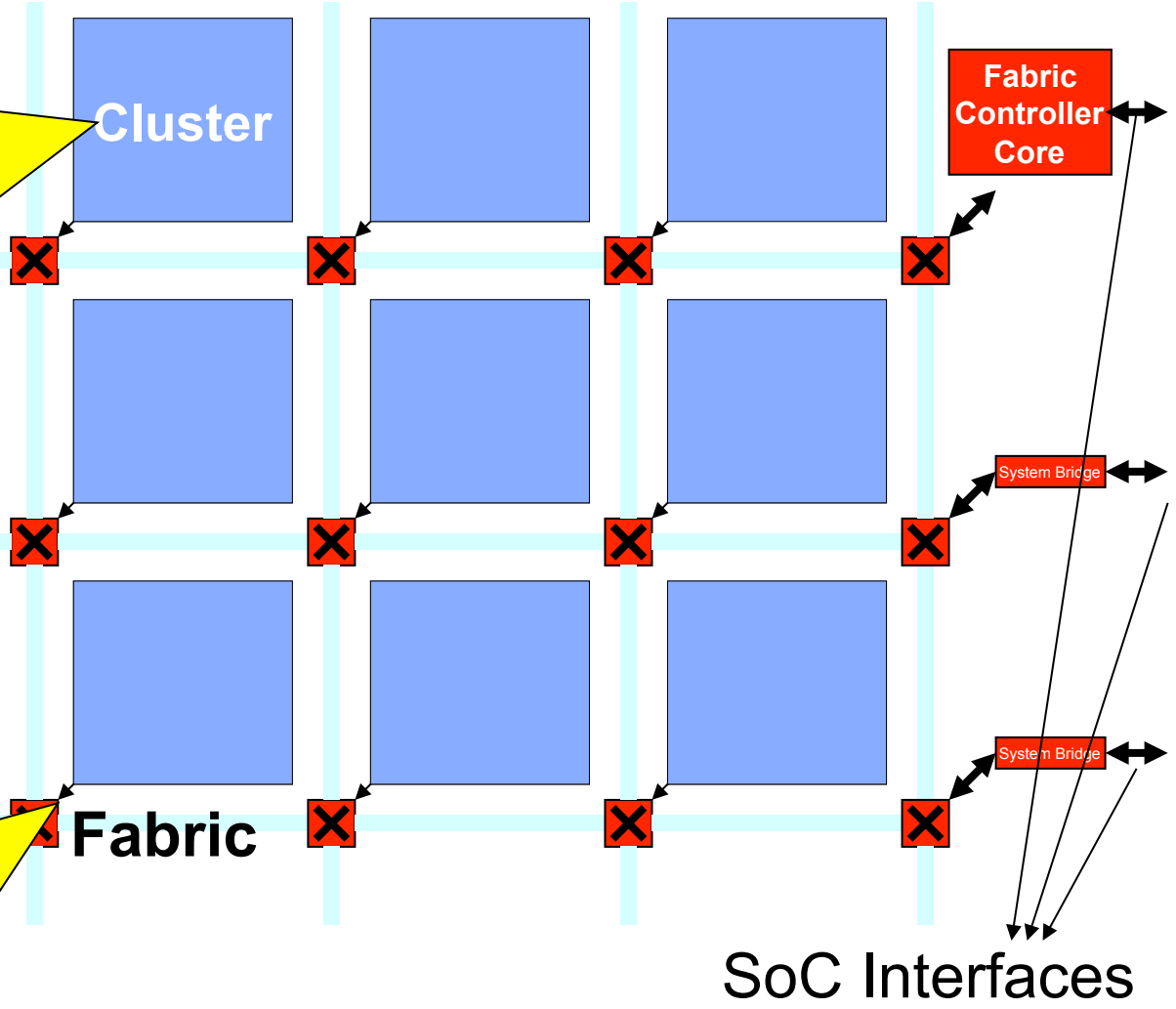


P2012 Top-Level Architectural View

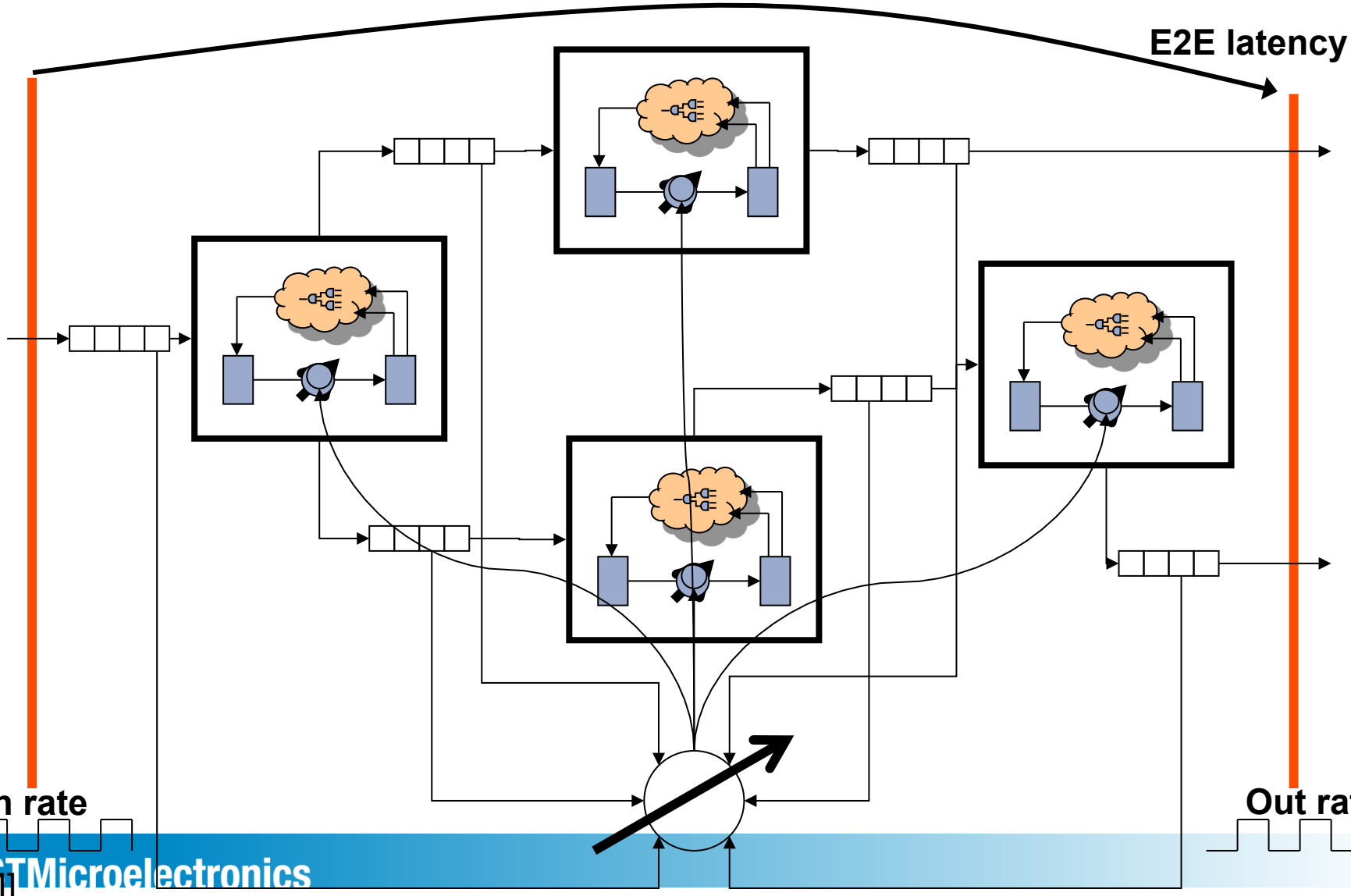
A tightly-coupled computing domain, sharing
- Local controller
- Memory
- Fabric interface

Strong physical & electrical decoupling (clock, power)

A loosely-coupled computing domain, sharing
- Asynchronous NoC
- Host interface –with configuration controller
- System bridges to L3 memory and other SoC mega-IPs



Modular, hierarchical Control



[STM]

Challenges

Silicon efficiency → Multi-Grained Acceleration

Silicon Efficiency @ Core Level

Instruction Level Parallelism

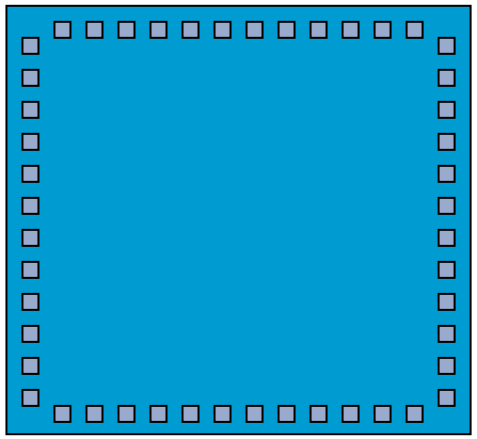
X=Y+Z || T=U*V

Data Level Parallelism

Vz[1..8]=Vx[1..8]+Vy[1..8]

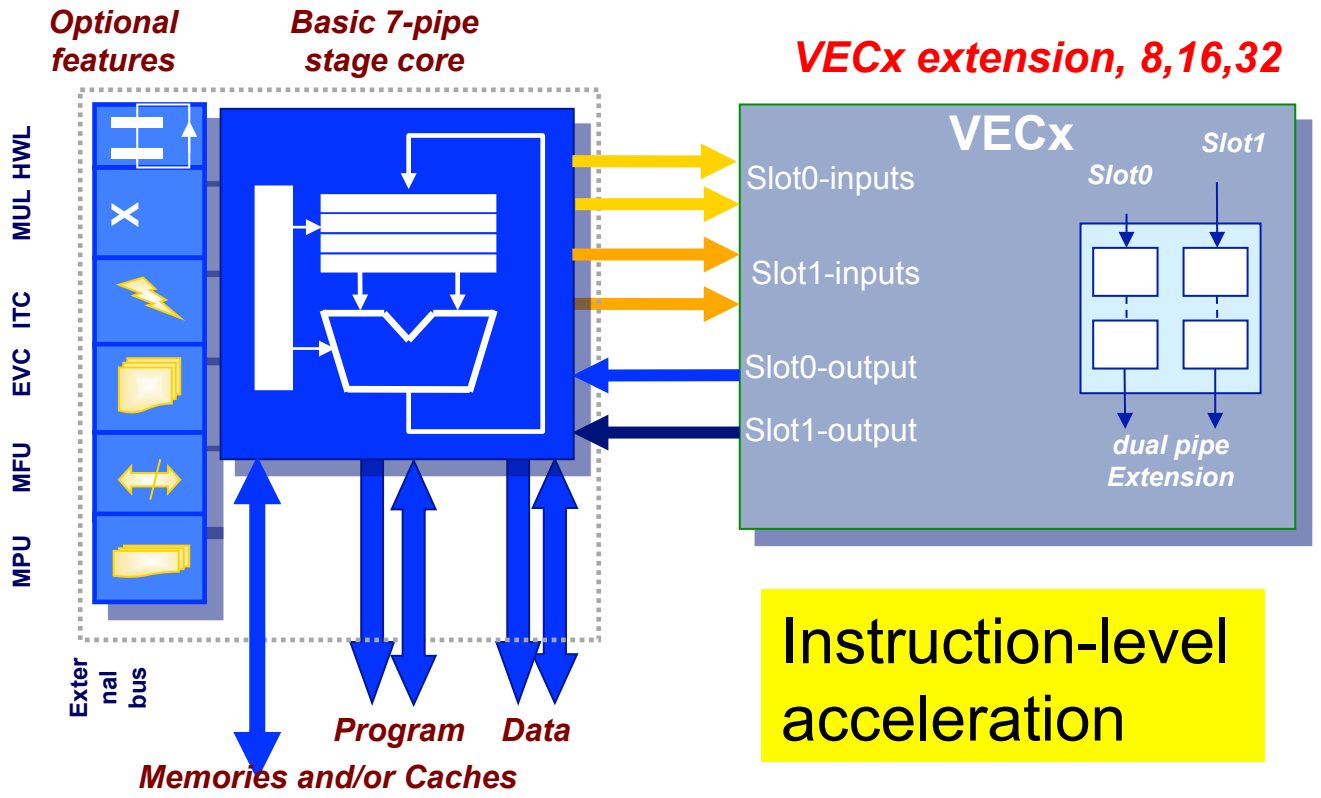
Instruction Optimization

<x1,..,xn> = f(……)



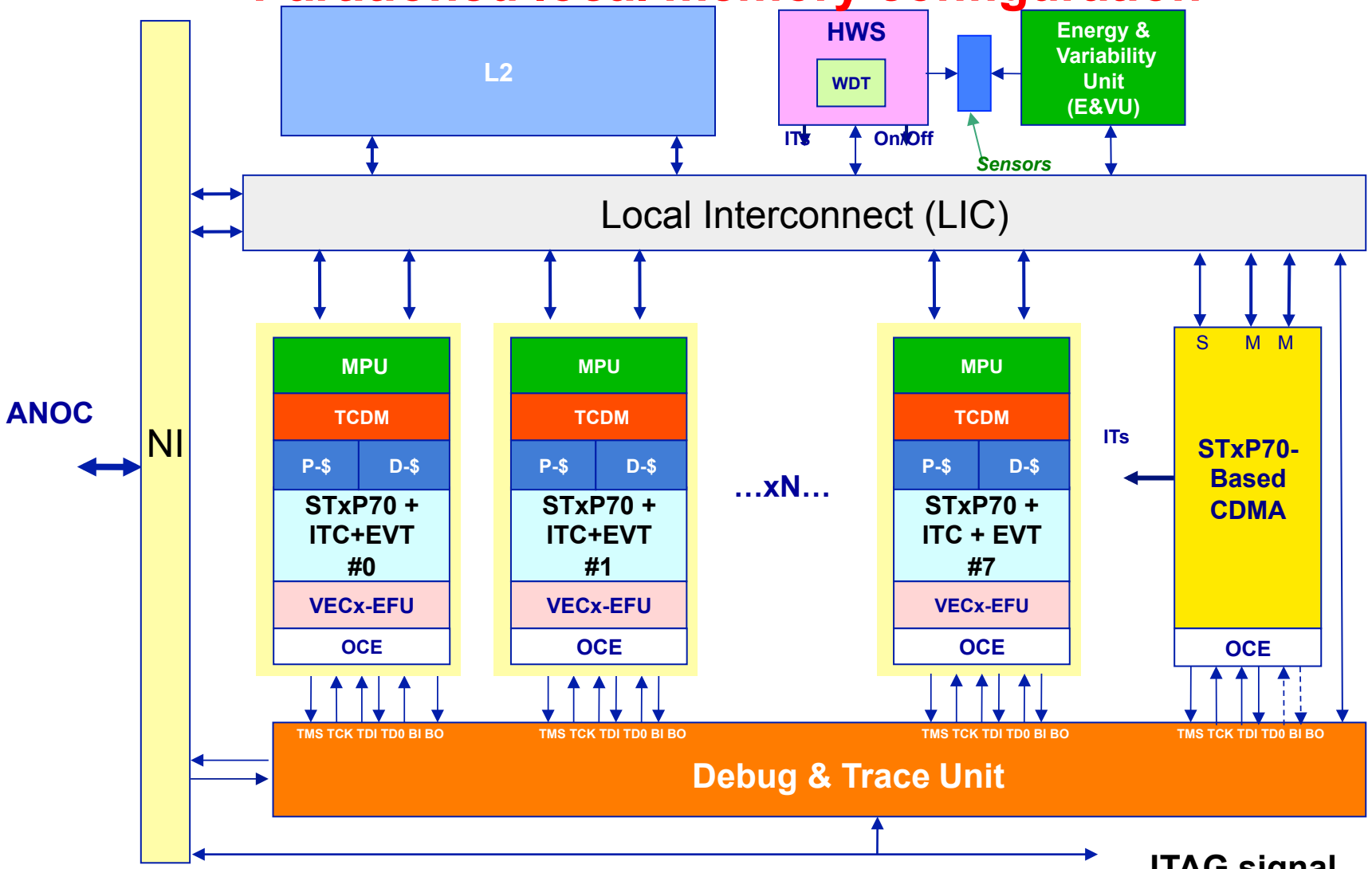
P2012 Core

- STXp70: an ASIP “on steroids”
  - Domain-specific instructions
  - Domain-specific parameterized vector extension ***VECx***
  - Dual issue



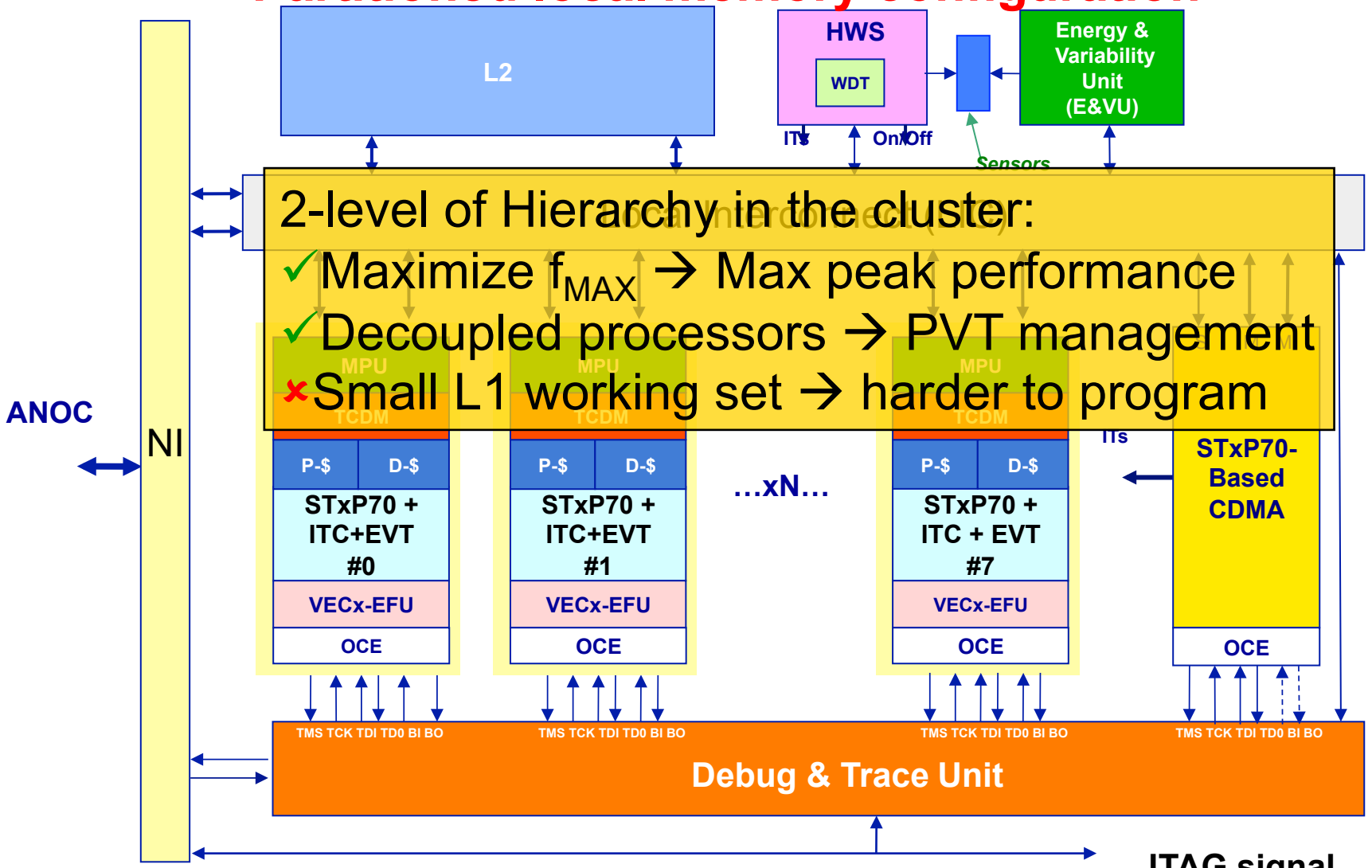
Multi-processor Cluster

Partitioned local memory configuration



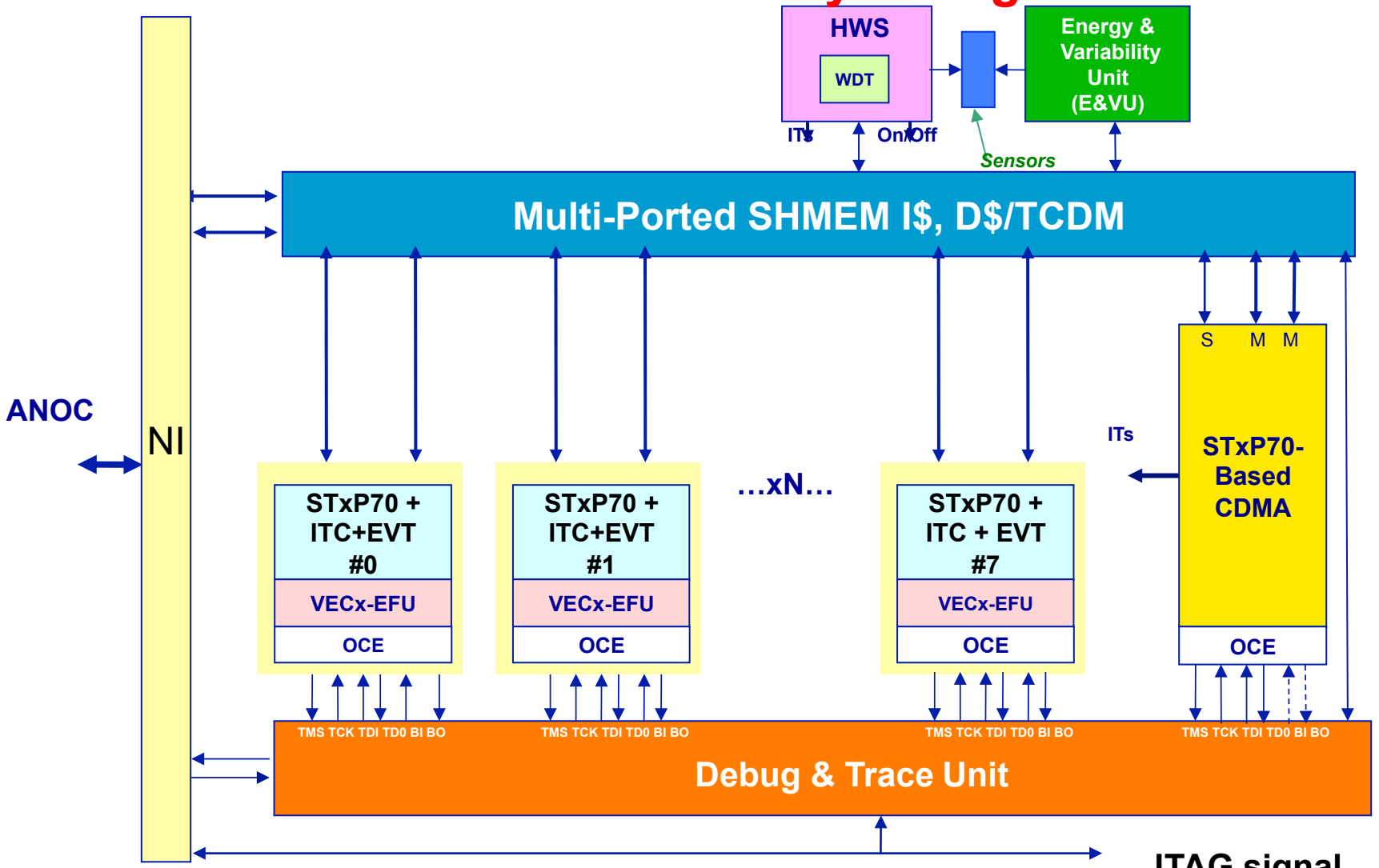
Multi-processor Cluster

Partitioned local memory configuration



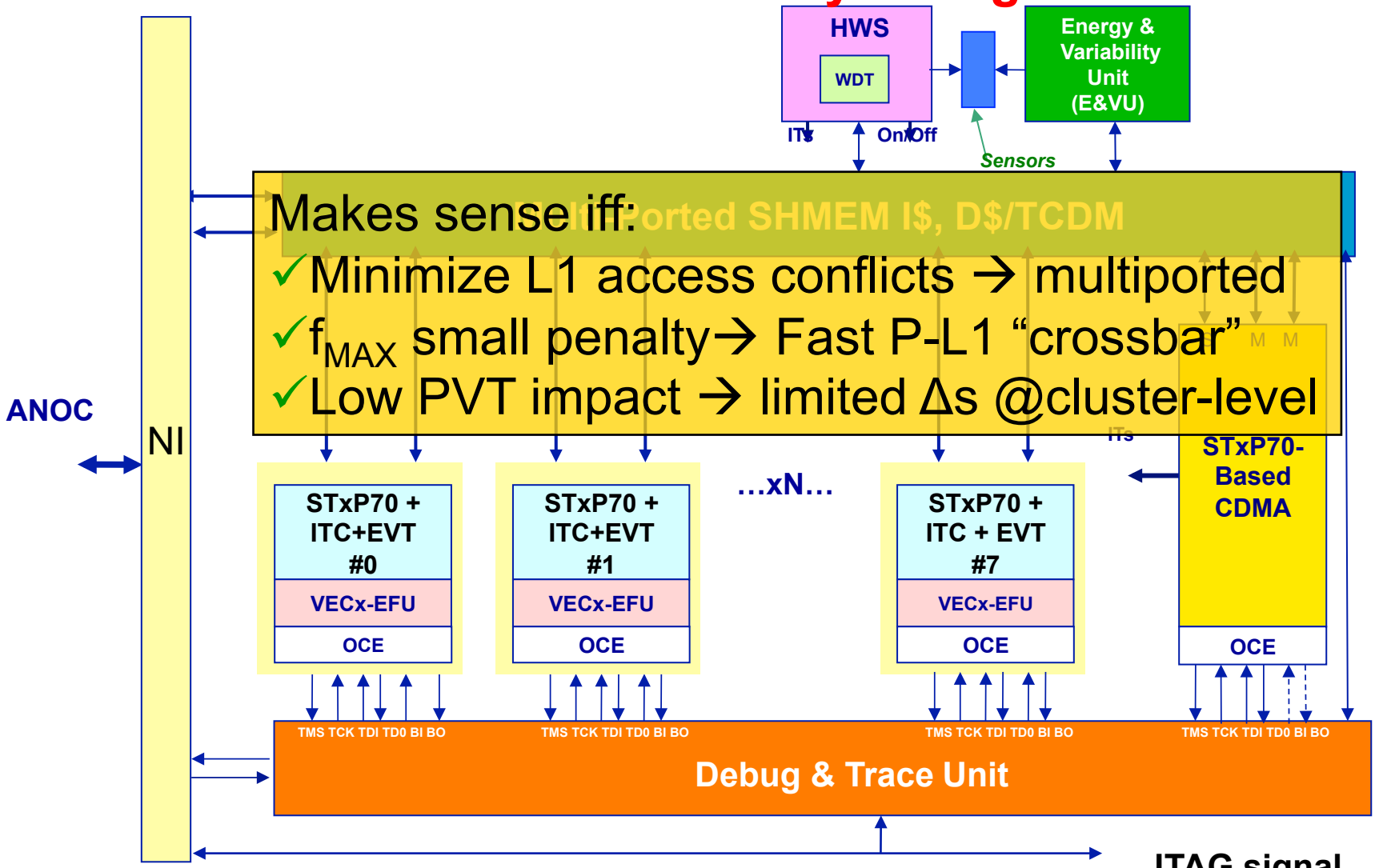
Multi-processor Cluster

Shared local memory configuration

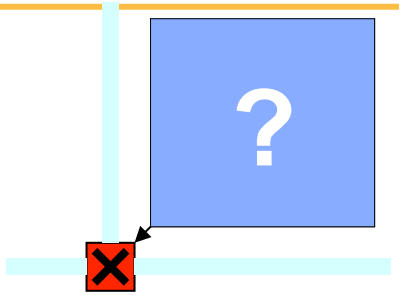


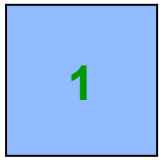
Multi-processor Cluster

Shared local memory configuration

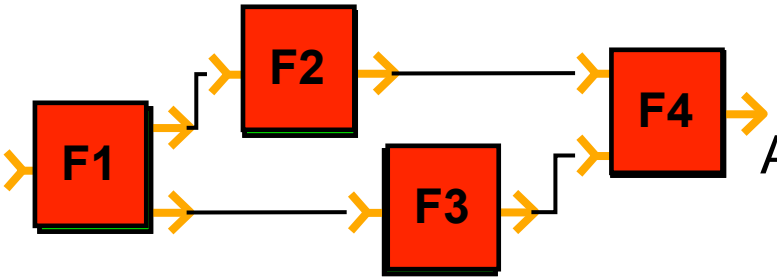


Efficient? Still not enough!

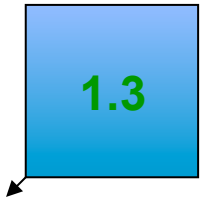




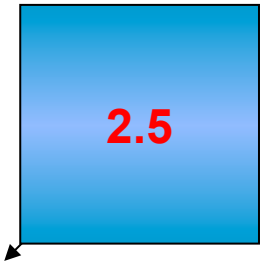
Pure HW custom block



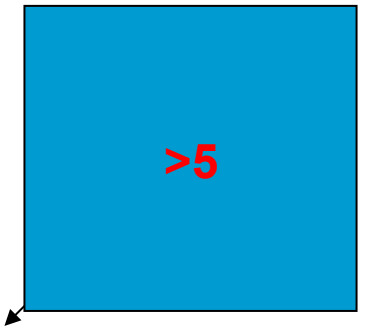
Application spec



P2012 HW cluster



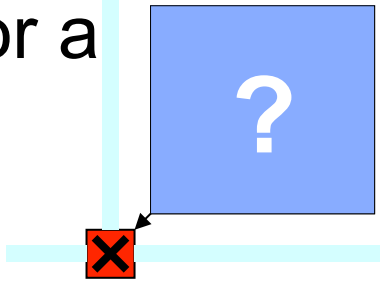
P2012 Heterogeneous HW-SW cluster



P2012 Homogeneous SW cluster

Efficient? Still not enough!

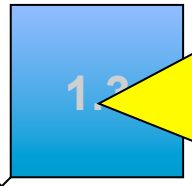
- Goal: provide cluster-level infrastructure for a range of mixed HW-SW implementation options



Pure HW

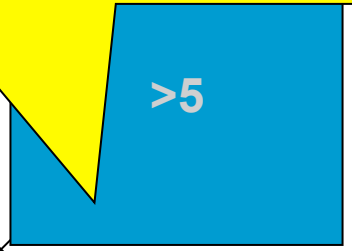
Why?
- SW flexibility is maintained
- Leverage pre-existing application-specific processor IPs
- Support for mixed Proc+HW accelerated function within cluster
- SW componentization + high-level specification & programming environment available

Why?
- Fastest to silicon
- Max regularity (yield)
- Max flexibility
- IP development, integration and qualification
- Demonstrate P2012: silicon, programming model and tools



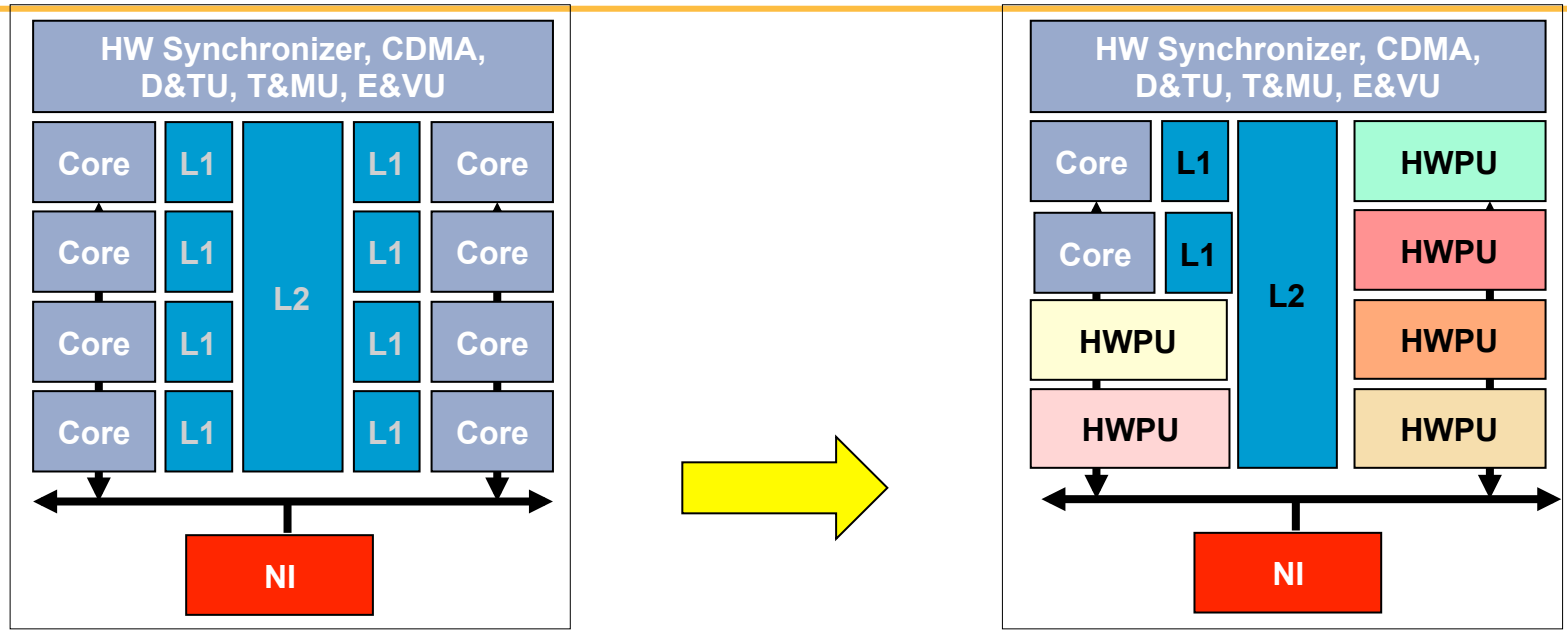
P2012 HW

Why?
- Minimum extra cost
- Max energy efficiency
- Scalability and modularity are preserved
- Infrastructure handles communication and synchronization



P2012 Homogeneous SW cluster

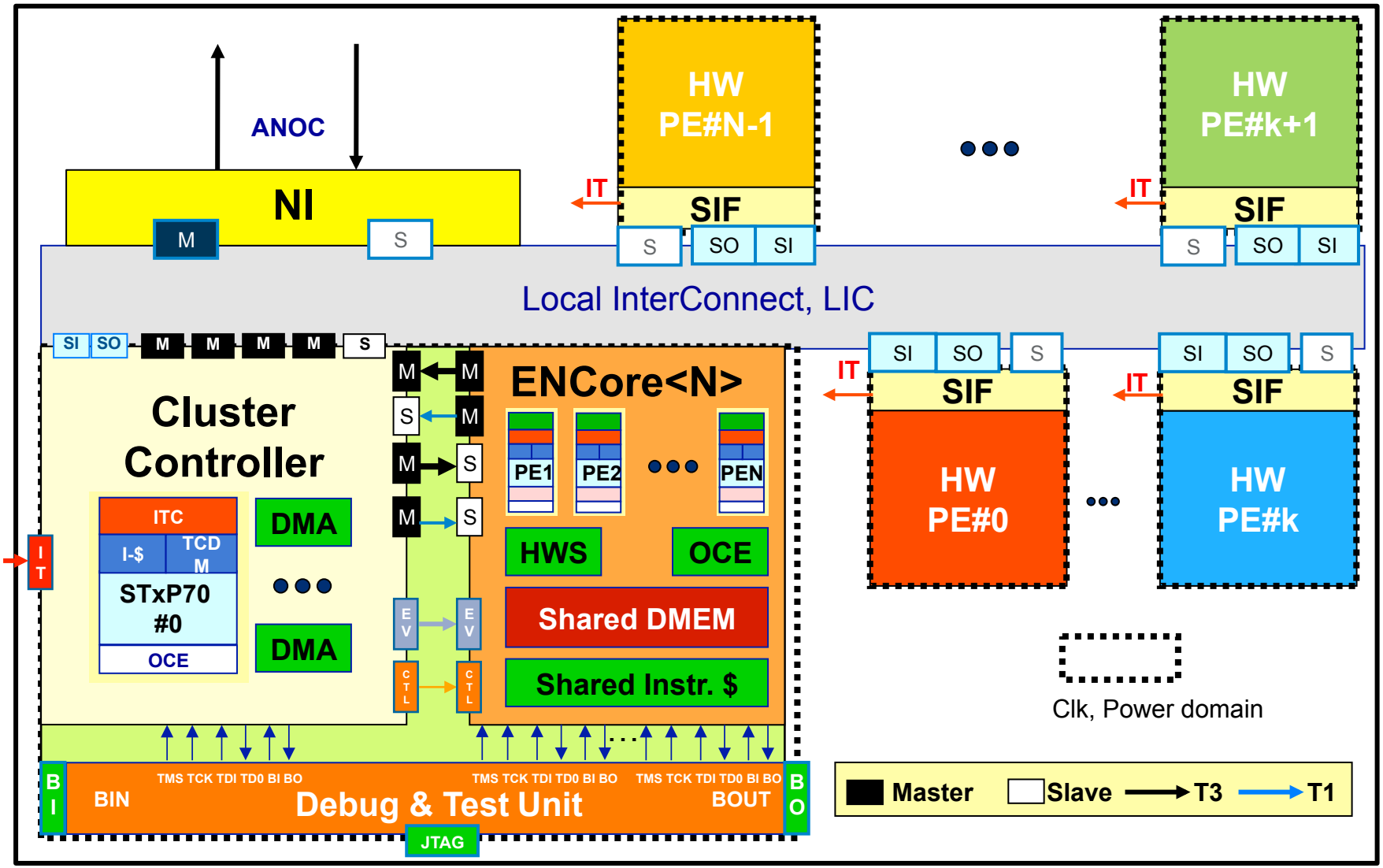
P2012 Heterogeneous Cluster



- Homogeneous tiles
- Programmable cores only
- Energy & Variability management
- Asynchronous interconnect
- Multi-Proc design

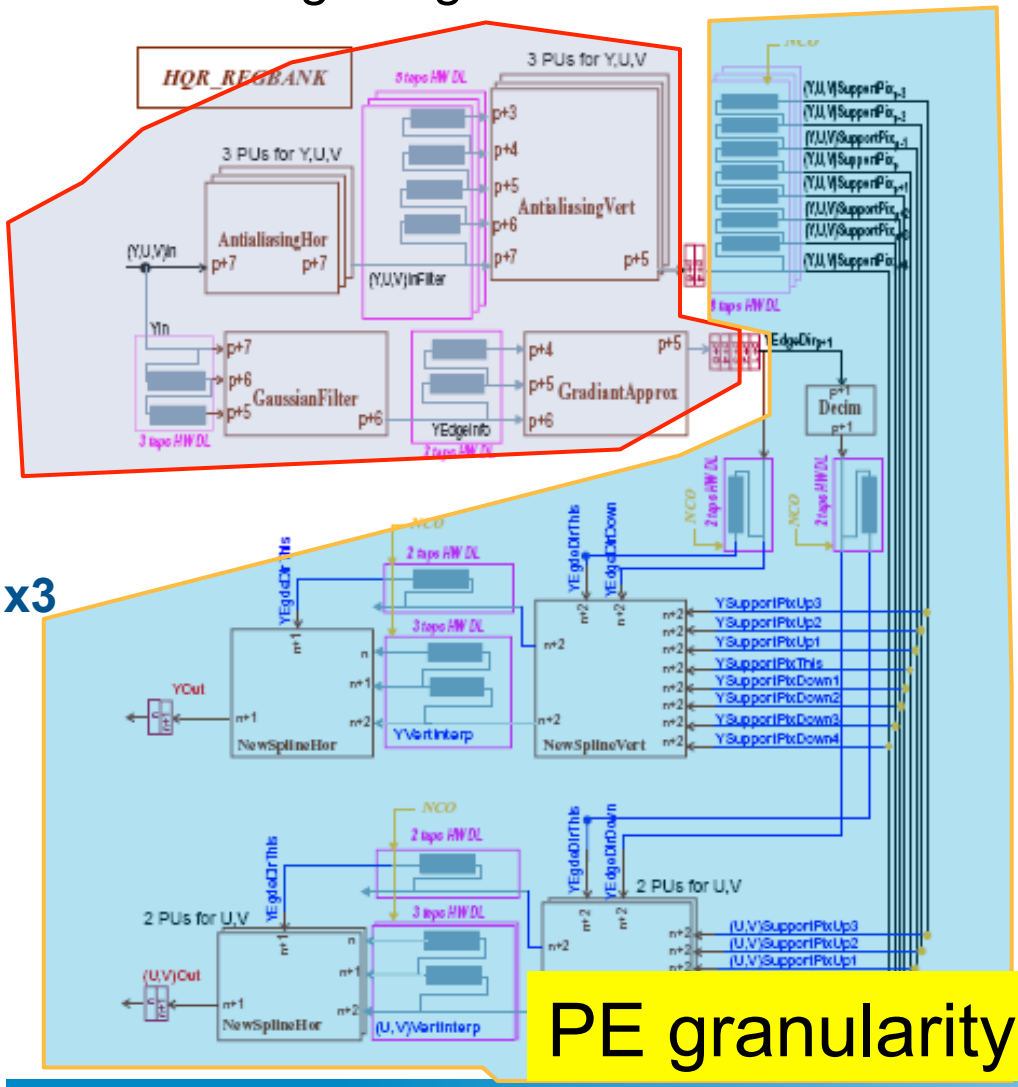
- Heterogeneous tiles
- Mix of programmable cores and hardware blocks within a tile
- Energy & Variability management
- Asynchronous interconnect
- System Level design

P2012 HW-SW Cluster

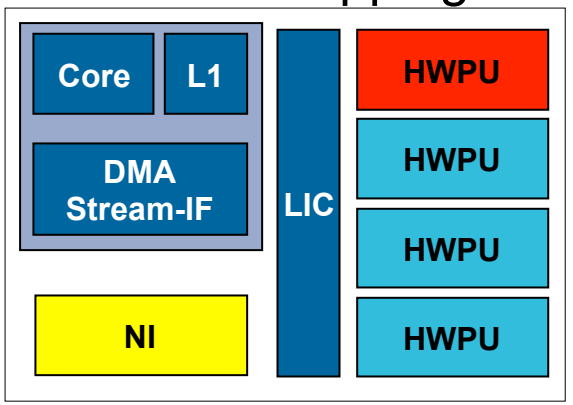


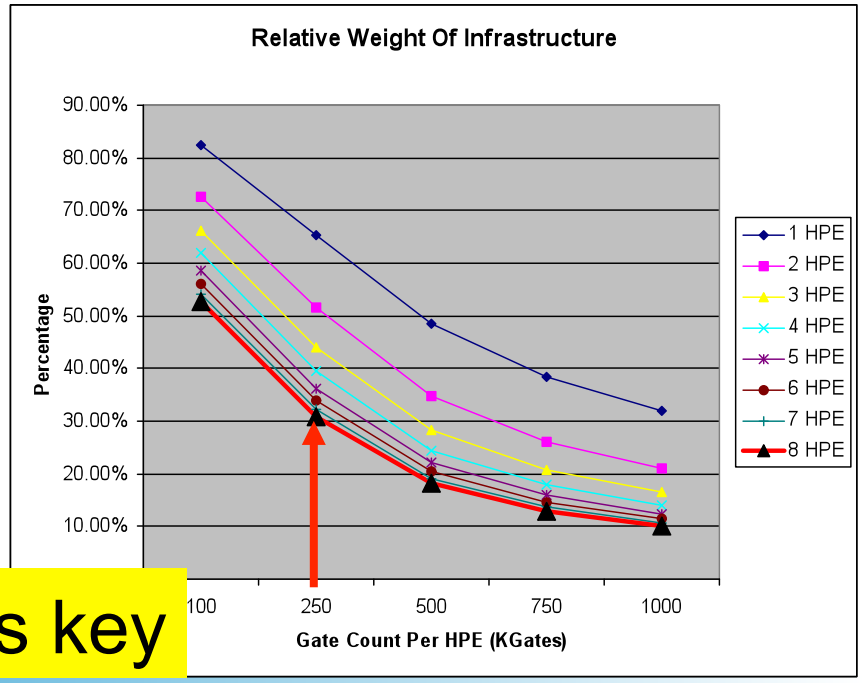
What is the cost of infrastructure?

Pre-existing Image Enhancement HW-IP



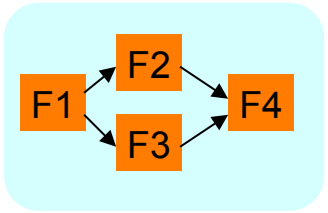
P2012 Mapping



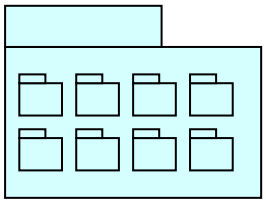


PE granularity is key

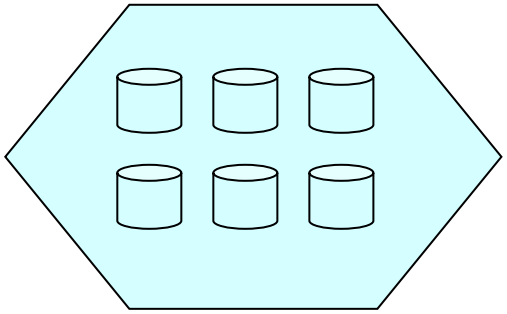
P2012 Design Flow



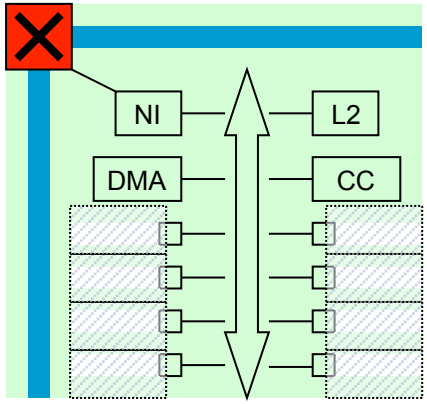
Application



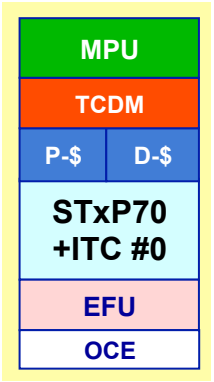
Runtime SW libraries & drivers



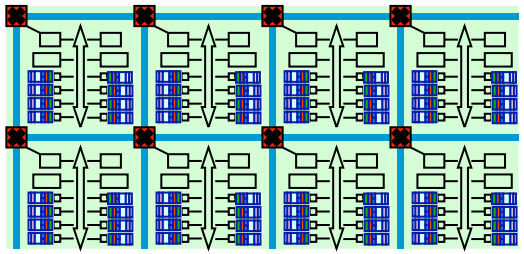
Tools



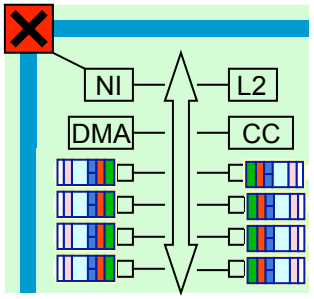
Customizable tile



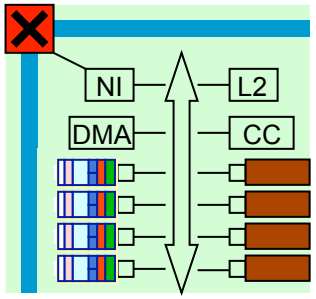
Customizable core



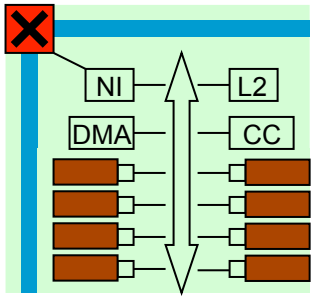
V1 SW fabric



SW only cluster



HW/SW cluster

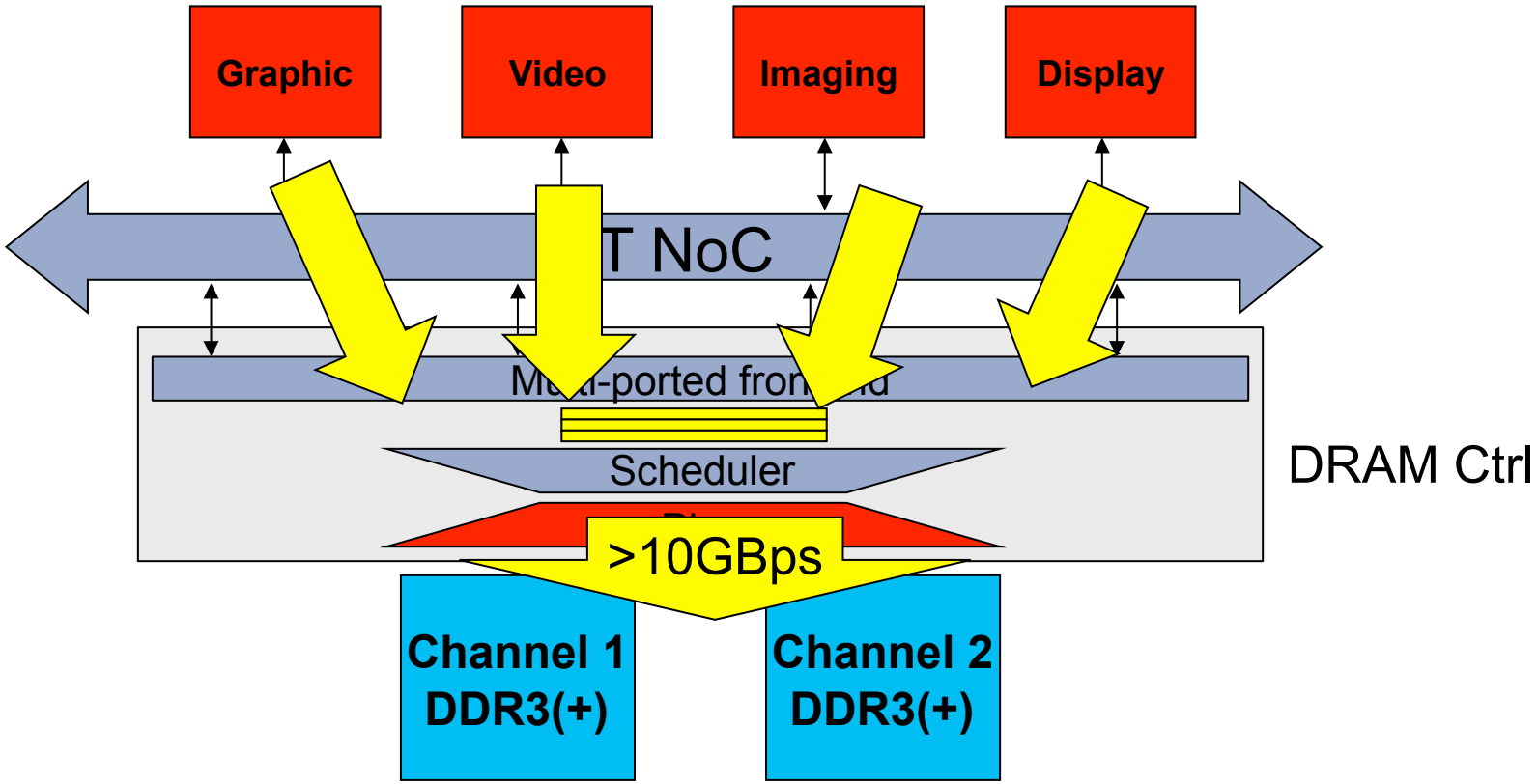


HW cluster

Challenges

Memory Bottleneck → 3D-Integrated Hierarchy

Next-Gen Mobile Platform Traffic

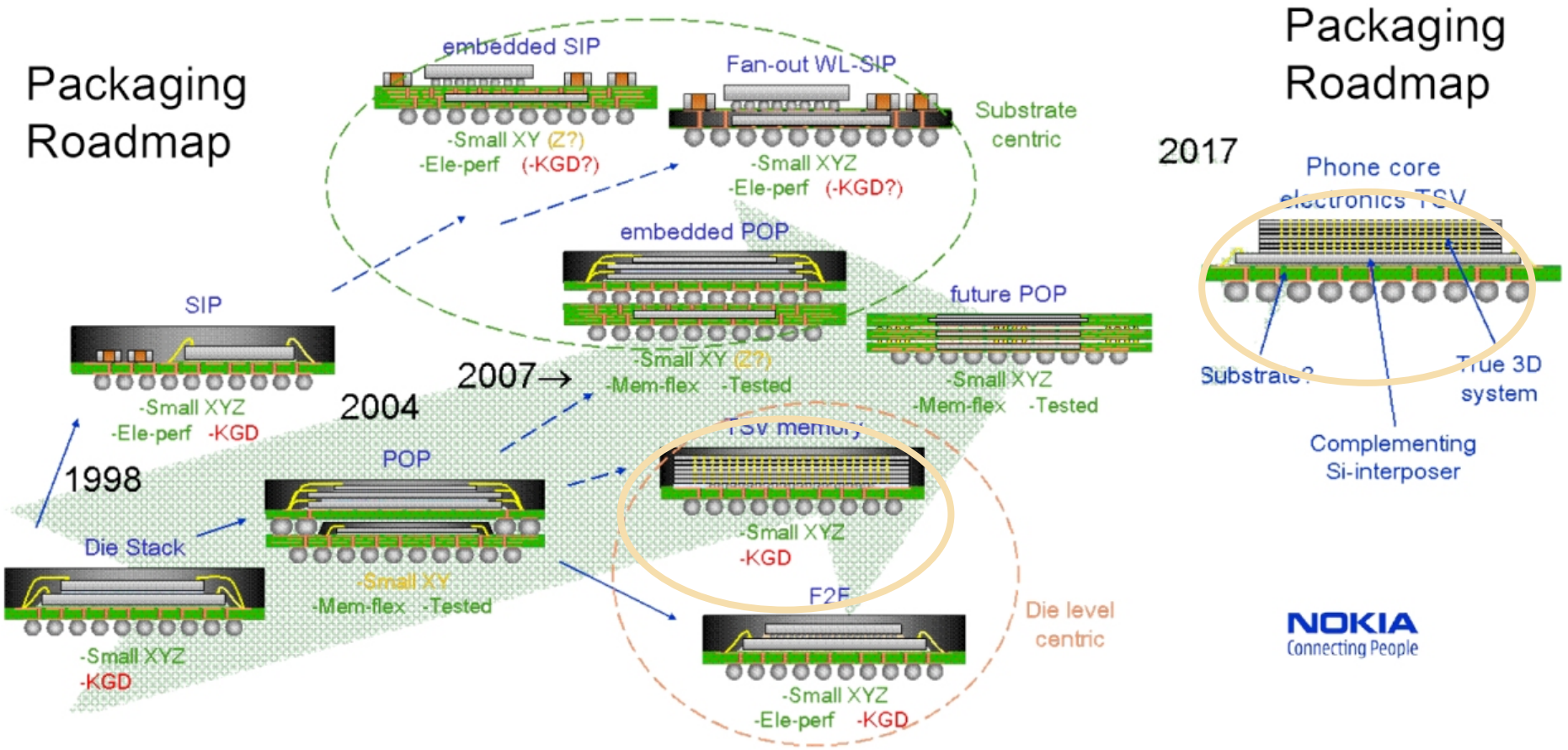


- Multiple *bandwidth hungry* initiators (Display, Graphic, Video, Imaging) that exchange data through main memory buffers
- Aggregate bandwidth is rapidly raising
- Traditional JDEC DRAM channels are saturating
- New standards are increasingly expensive (power, area, timing convergence)

31

Help from technology

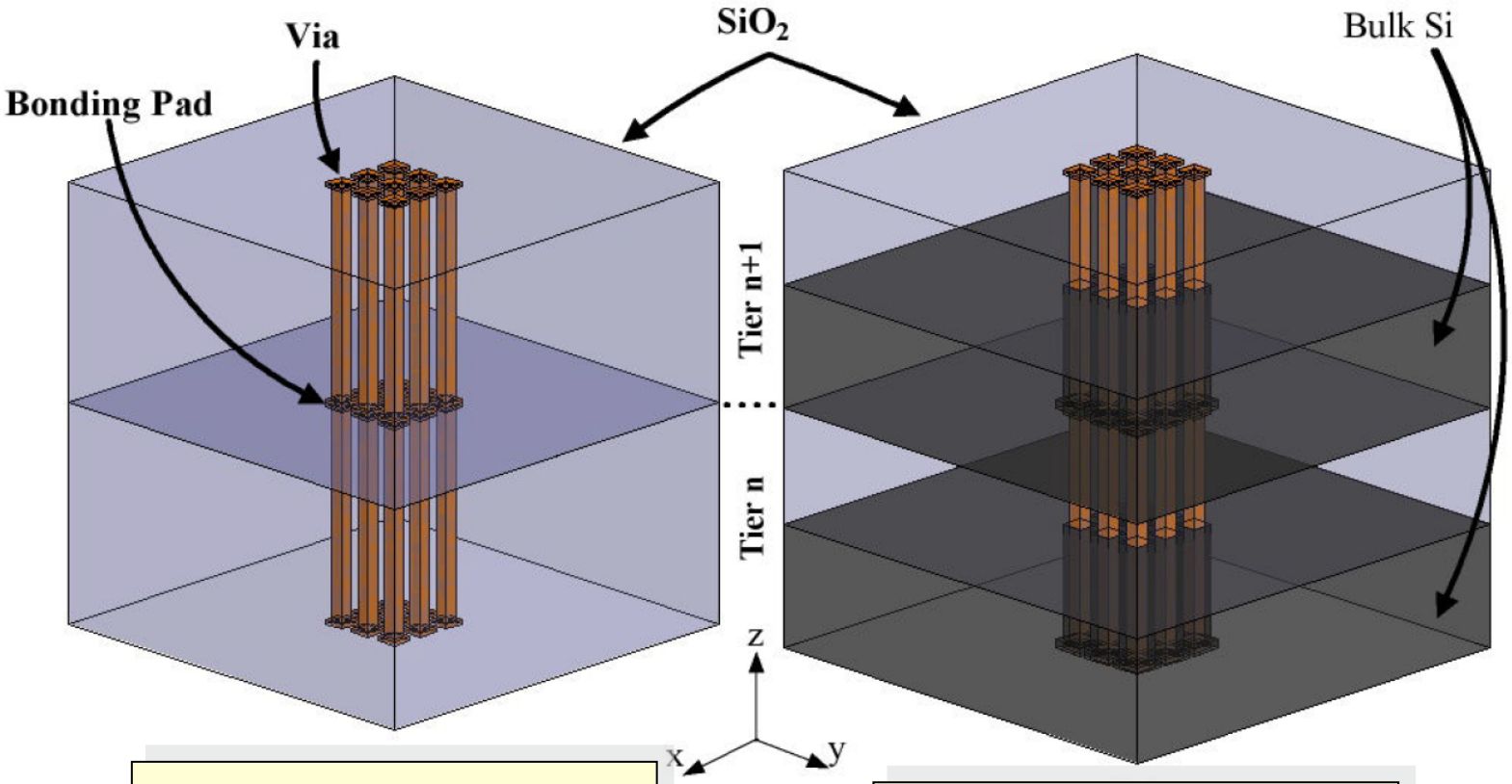
Bandwidth hungry, even more so for predictability



Through-silicon vias are at the technology bleeding edge today

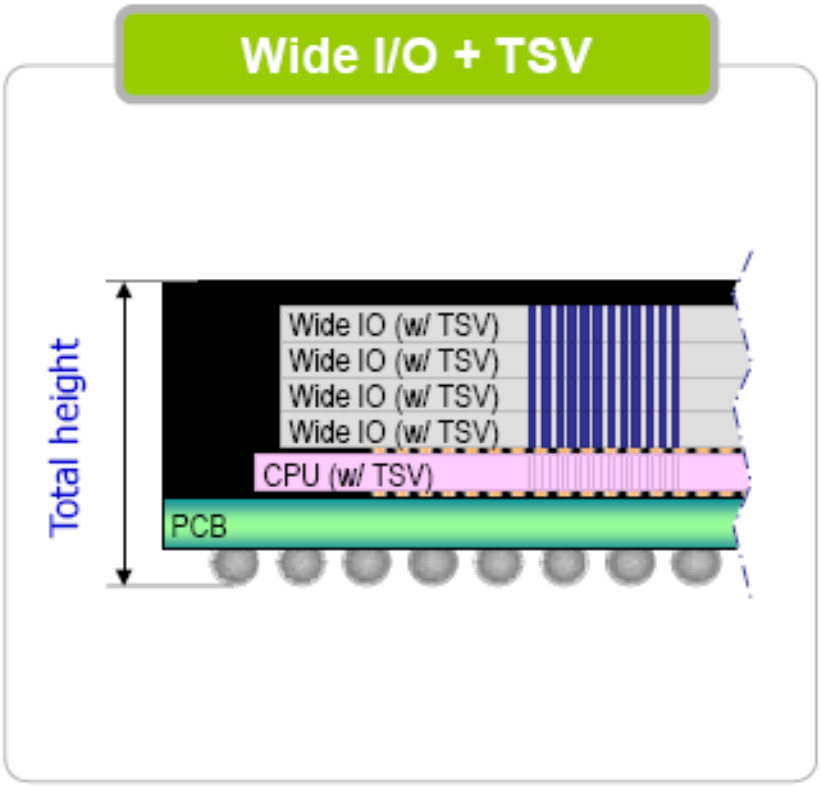
Industry interest is growing: http://www.emc3d.org/

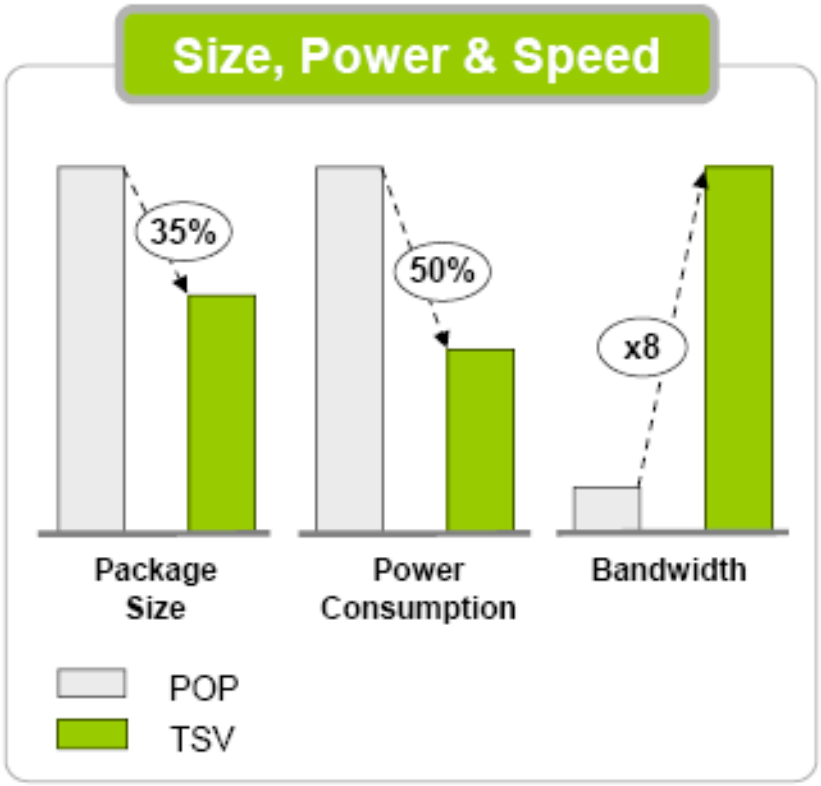
Understanding TSV technoology



- For a whole via of 50µm, delay is 16/18.5ps (SOI/bulk)
- For a 1.5mm horizontal link, delay is around 200ps

3D DRAM+Logic Integration



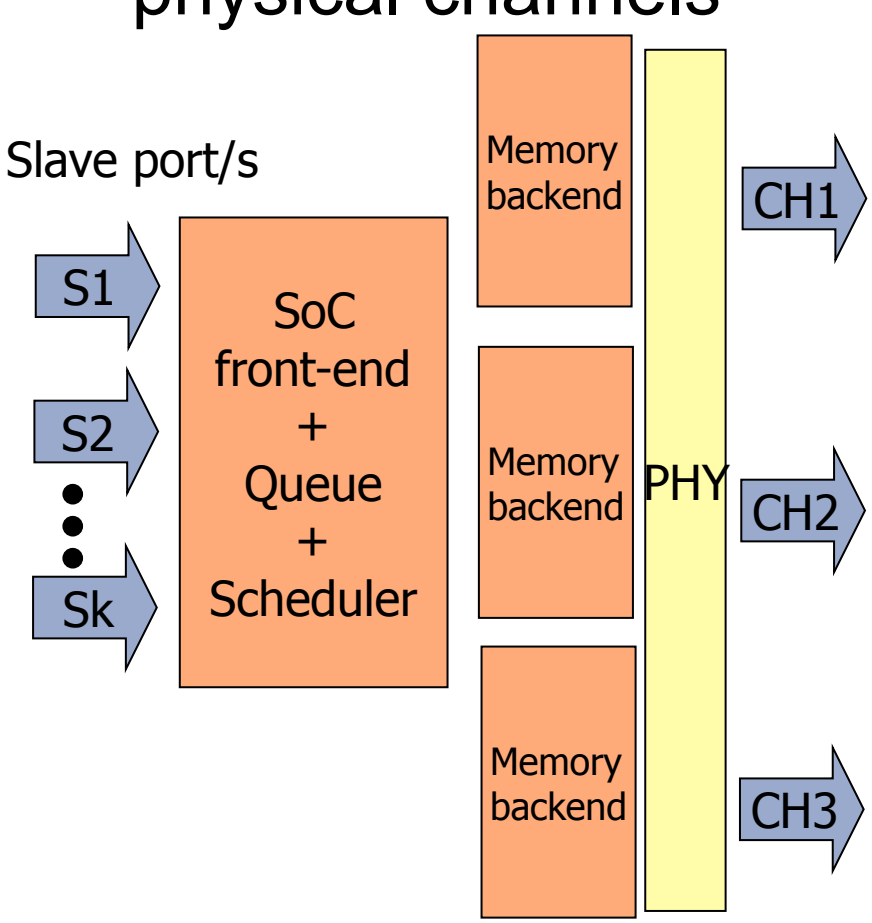


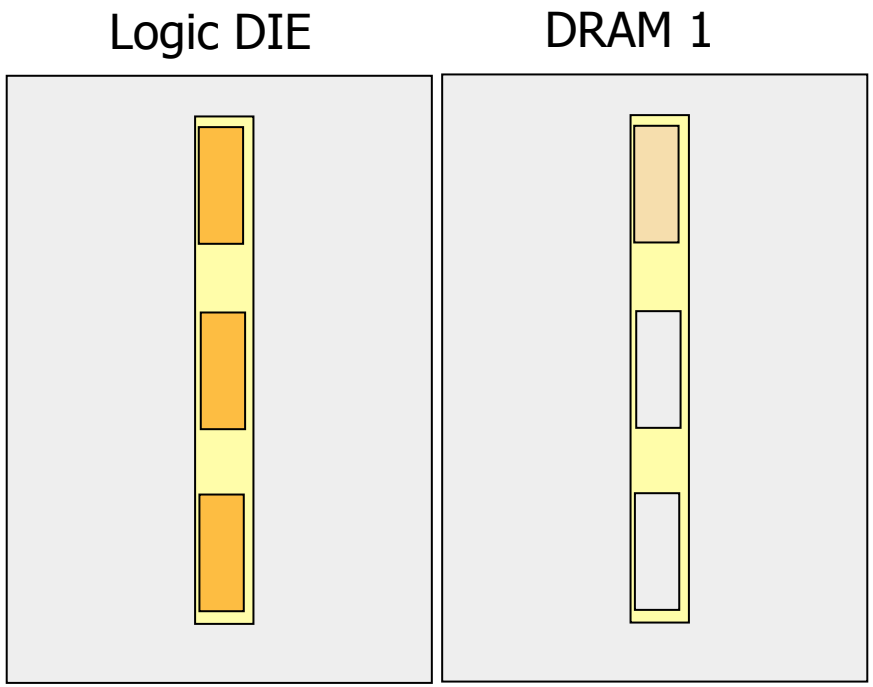
[Samsung10]

[Samsung09]

3D-Multi-Channel Wide-IO Interface

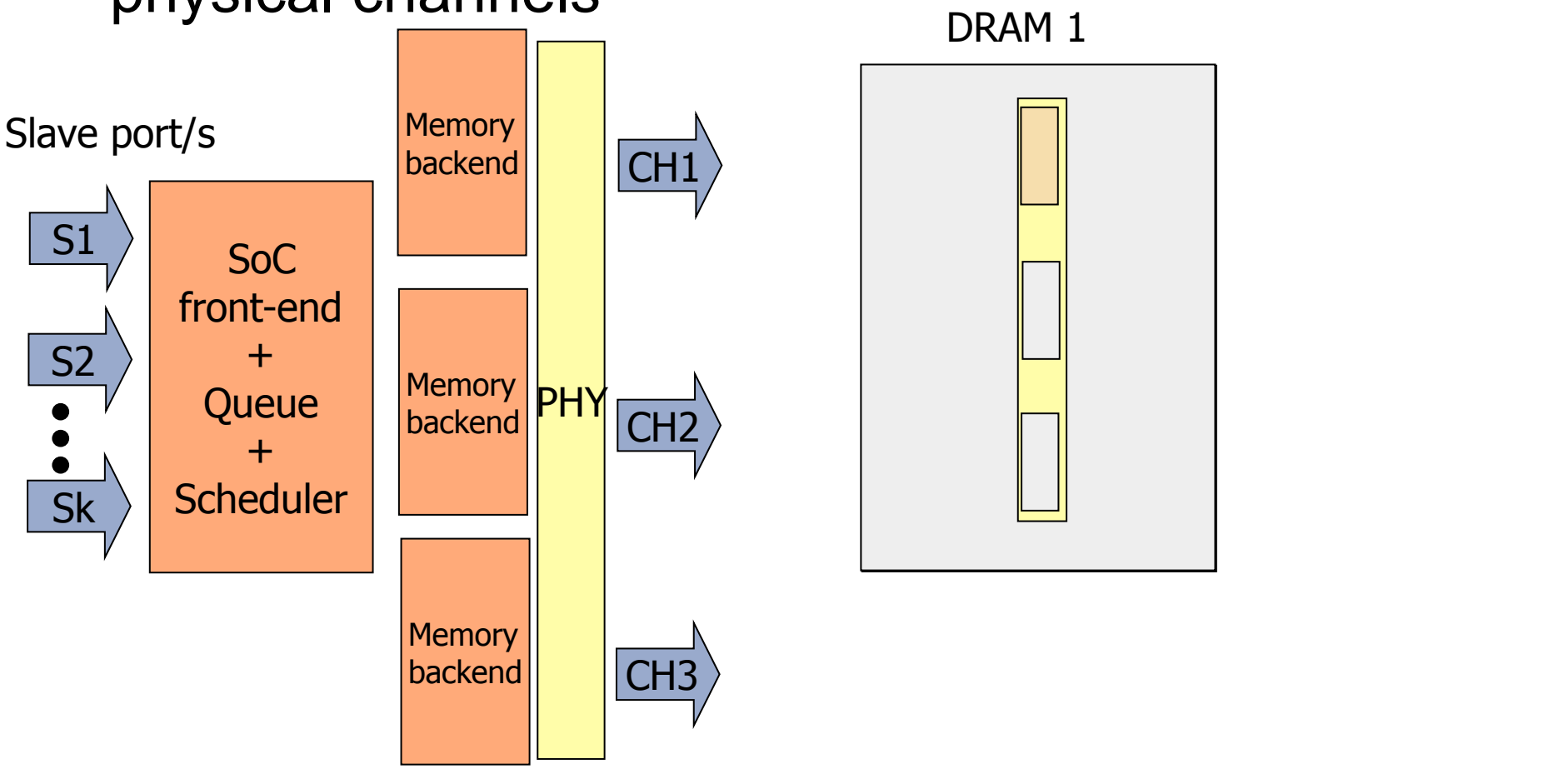
- Multiple logical channels onto multiple vertical physical channels





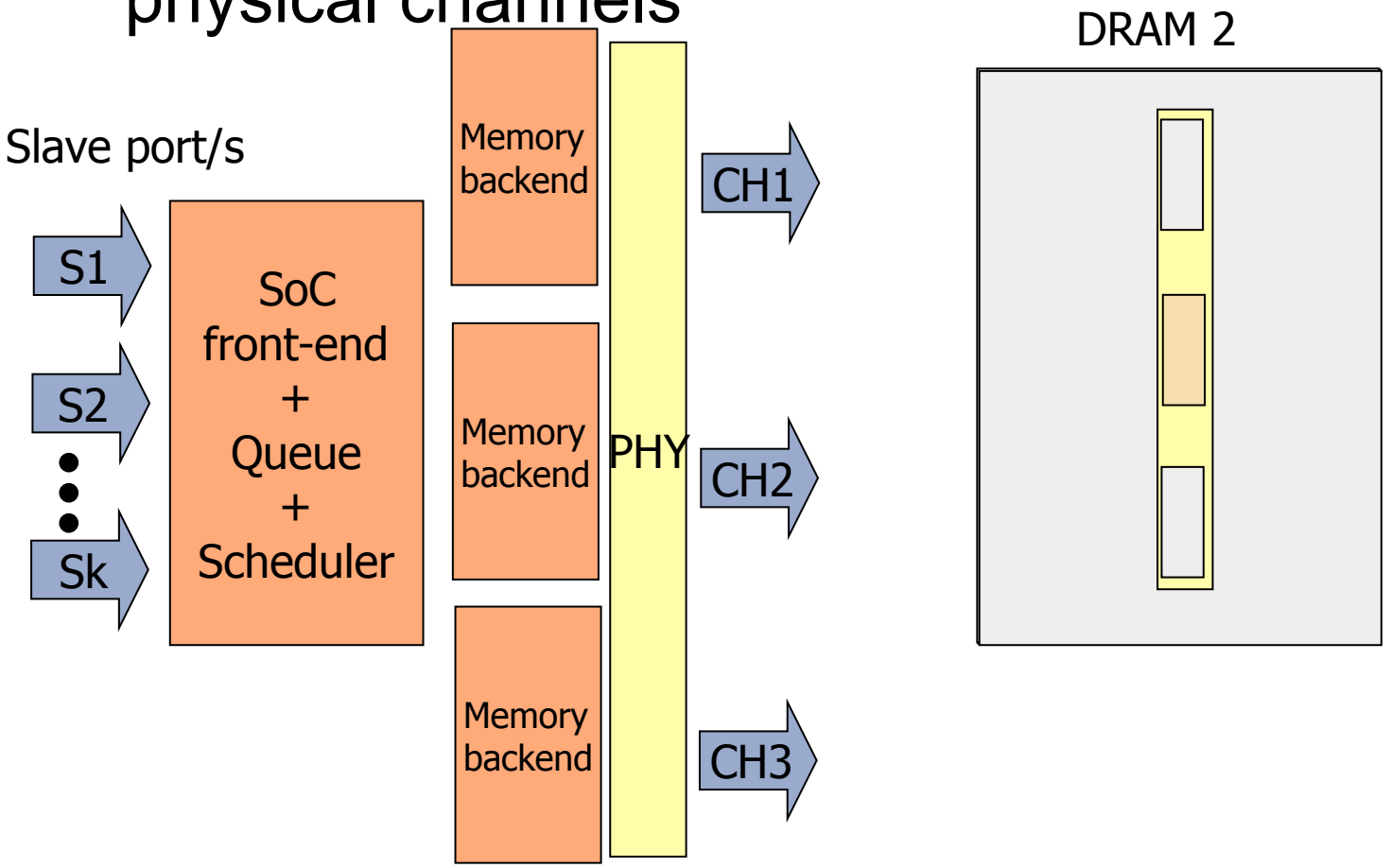
3D-Multi-Channel Wide-IO Interface

- Multiple logical channels onto multiple vertical physical channels



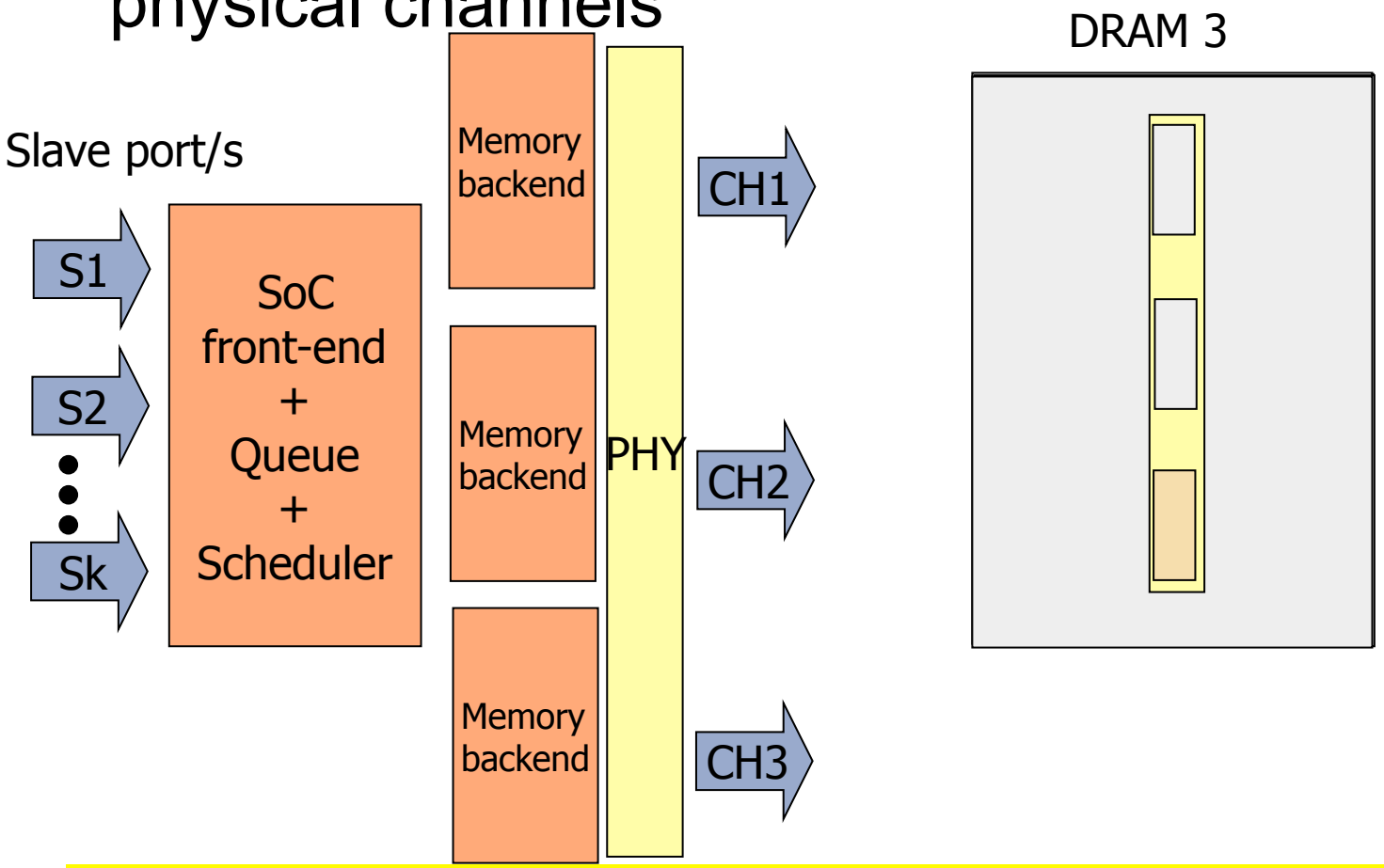
3D-Multi-Channel Wide-IO Interface

- Multiple logical channels onto multiple vertical physical channels



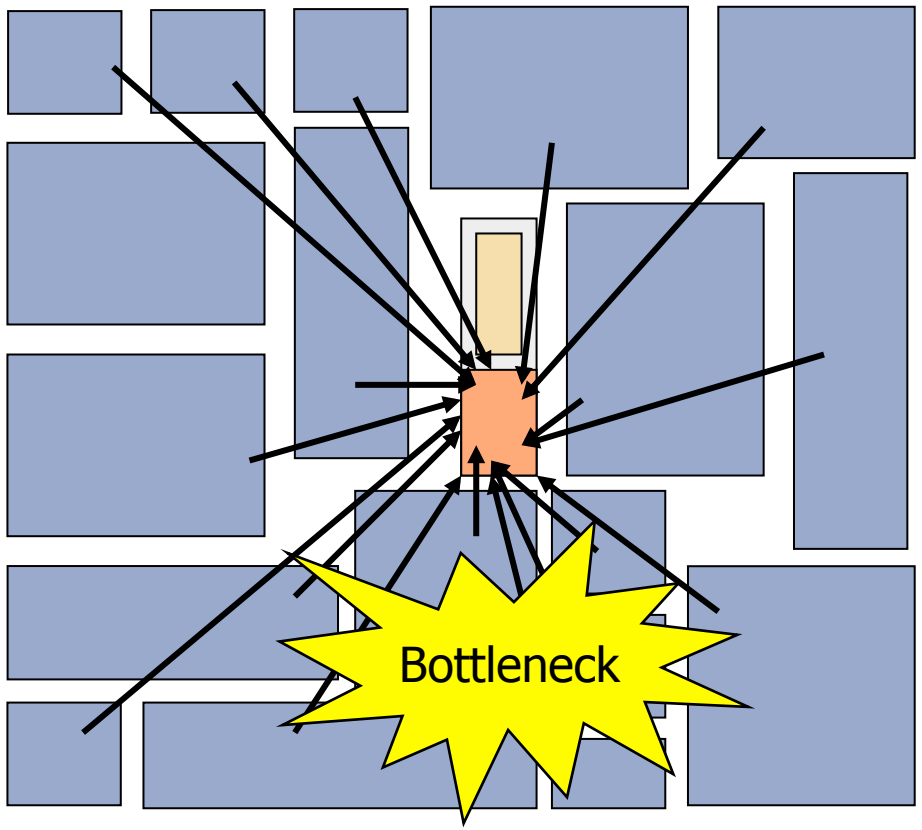
3D-Multi-Channel Wide-IO Interface

- Multiple logical channels onto multiple vertical physical channels



Advantage: does not require functional changes to DRAM interface

Scalability bottleneck

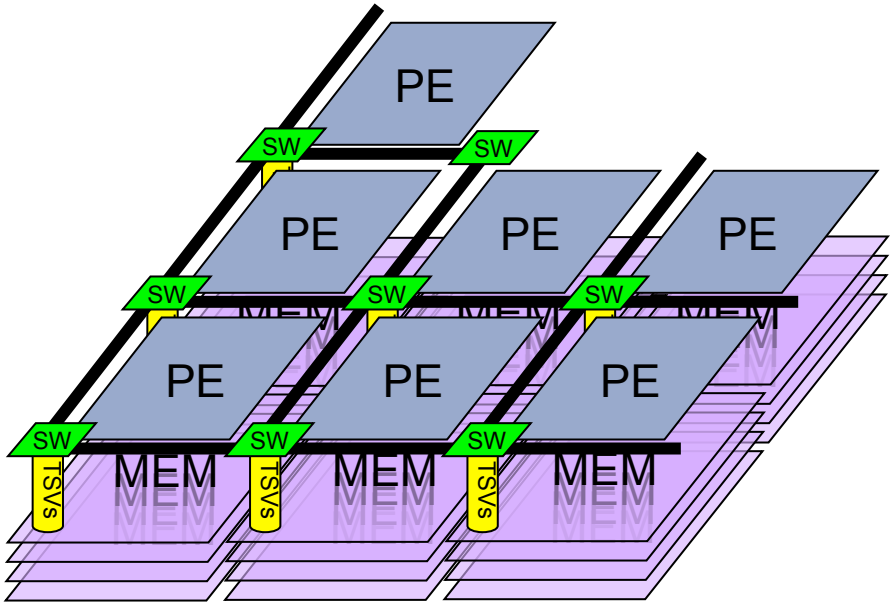


- Single controller becomes a bottleneck even with many slave ports
  - All cores need to reach it! Even using NoC interconnect, the latency price is high
  - Internal management of multiple slave interface and many transaction queues creates complexity bottlenecks
- This approach does not exploit the possibility of fine-grain distribution of 3D vias
- Creates a single point of failure

Scalable 3D-platform

3D-Network on-chip

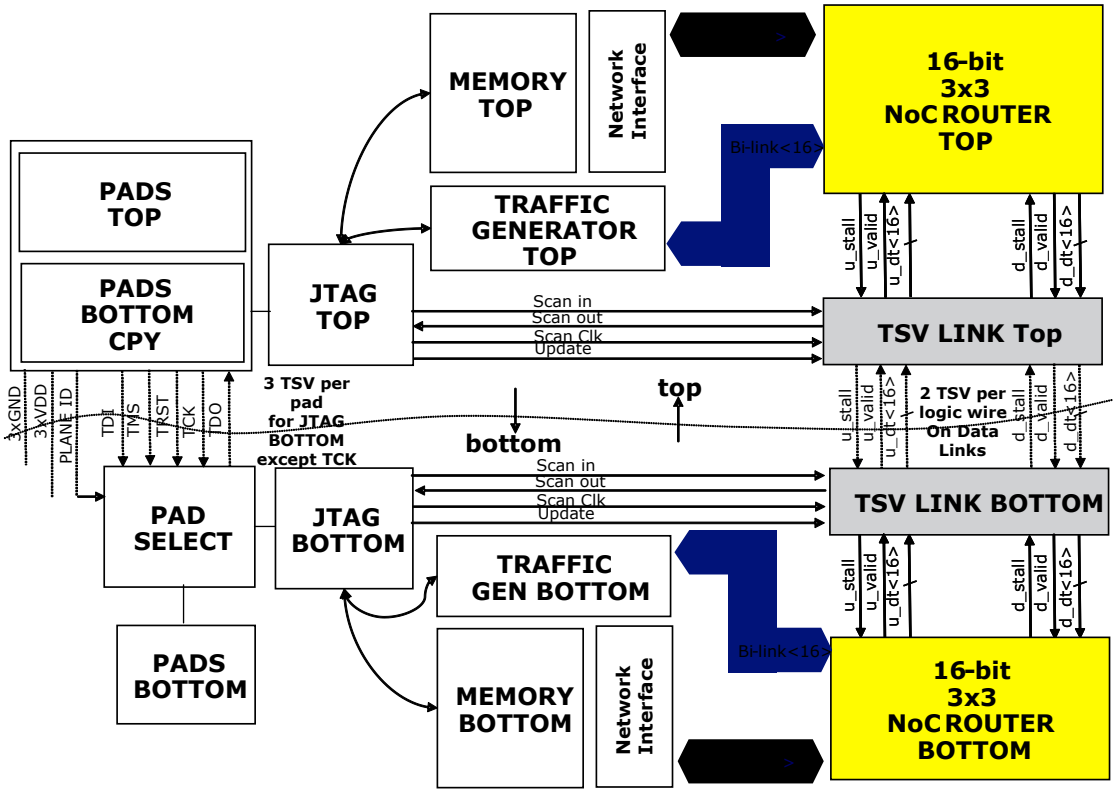
- Packet-based communication with QoS support (TDMA/priorities/ regulated traffic)
- Architecturally scalable: more nodes, more bandwidth
- Physically scalable: segmented P2P links



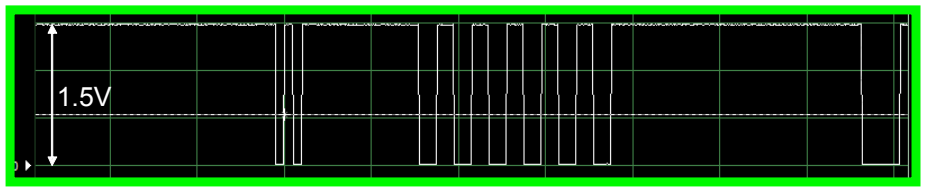
Vertically Integrated main memory (not only DRAM!)

- TSV main-memory communication from 10pJ/bit to 10fJ/bit
- 10^5 interconnect density increase

3D-NoC Implementation



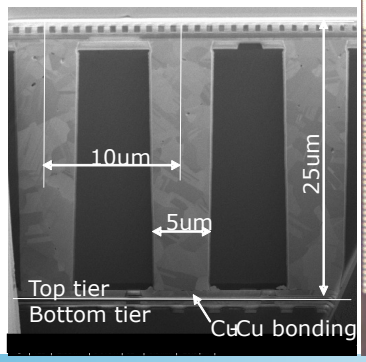
3D NOC Test Chip		
IMEC 130nm/200mm/2 metal layers		
Gate count	2x60kGates	
Chip Area	981umx1003um	
TSV Area of 3D NoC	700umx26um	
Simulated Power 2D traffic @1.2V/50Mhz/25C*	1.34mW	
Simulated Power 3D traffic @1.2V/50Mhz/25C *	1.39mW	
Simulated 3D Link Performance		
Energy/Delay 3D link@1.2V (combinational logic between as in Figure 3)	2pJ/bit	183 ps

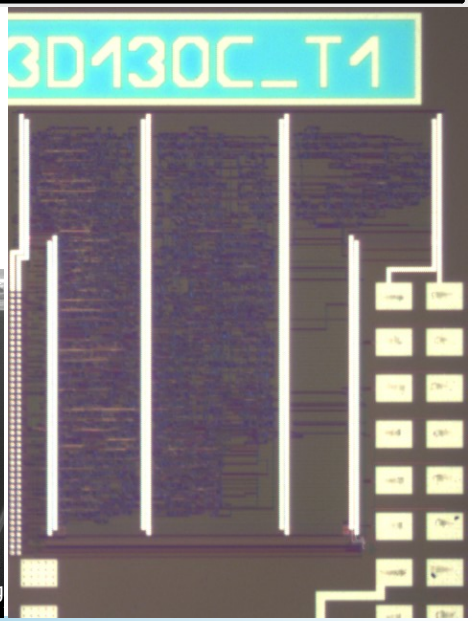


25Mhz@0.4-1.5V/25C

Limited by test equipment

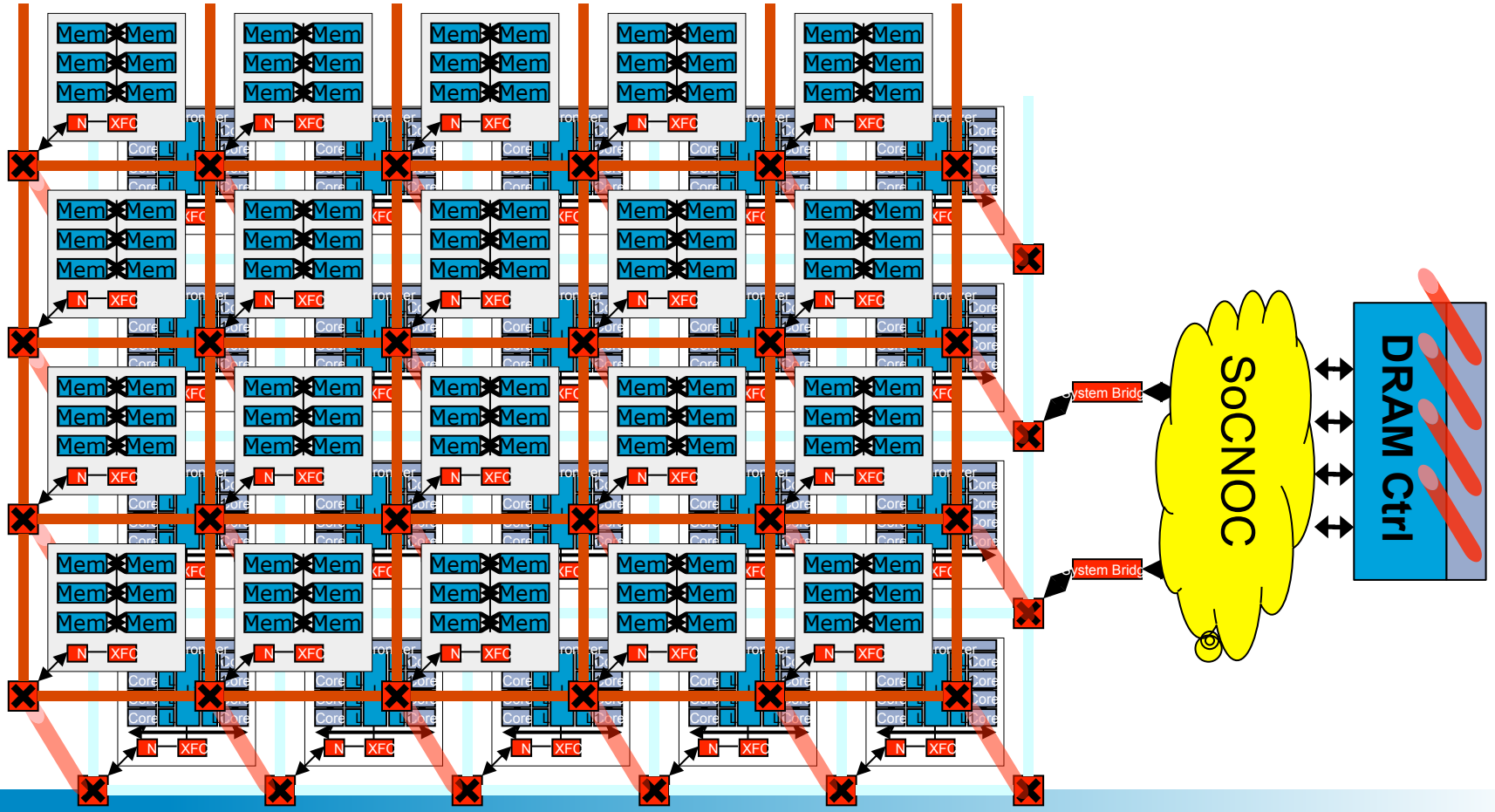
Measured TSV resistance	~20mOhm
Measured TSV capacitance	37fF(depl.)





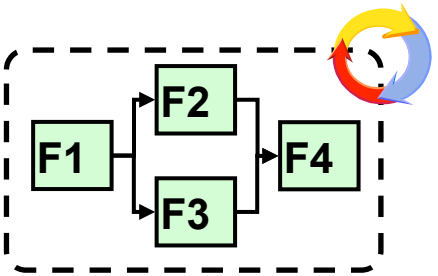
P2012 & 3D Integration

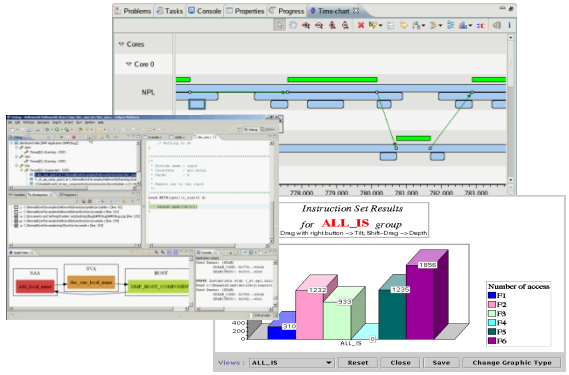
- Main memory →TSV-based DRAM wide-IO
- On-chip memory → 3D-NoC



Programming P2012

Programming Tools





- Programmer's environment for
  - High-level application capture and simulation
  - User-driven platform mapping
  - Programming model aware Debug, Trace and Analysis tools
- Leverages
  - Fabric Runtime
  - Host-side Dynamic Deployment Environment
  - xP70 compile, link and debug tools
  - GUIs STWorkbench
- **Supports multiple Parallel Programming Models**

Integrated Development Environment (STWorkbench)

Application C functions

Application Specification

Data Level Parallelism

Threading

Streaming

Apex

Patterns

Component repository

Leverage existing technologies

MIND Component Infrastructure

Task-to-PE Assignment

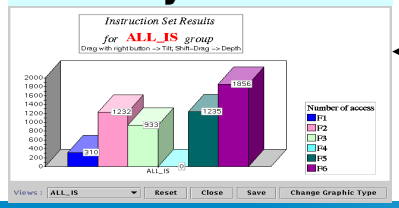
Communication Generation

Component assembly, C code generation

Runtime Execution Engines

Native Programming Layer

Performance Analysis



P2012 Platform

L3

Shared L1

DMA

HWS

PE0

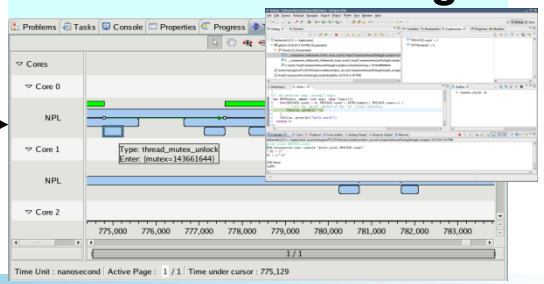
PE1

...

PEn

C

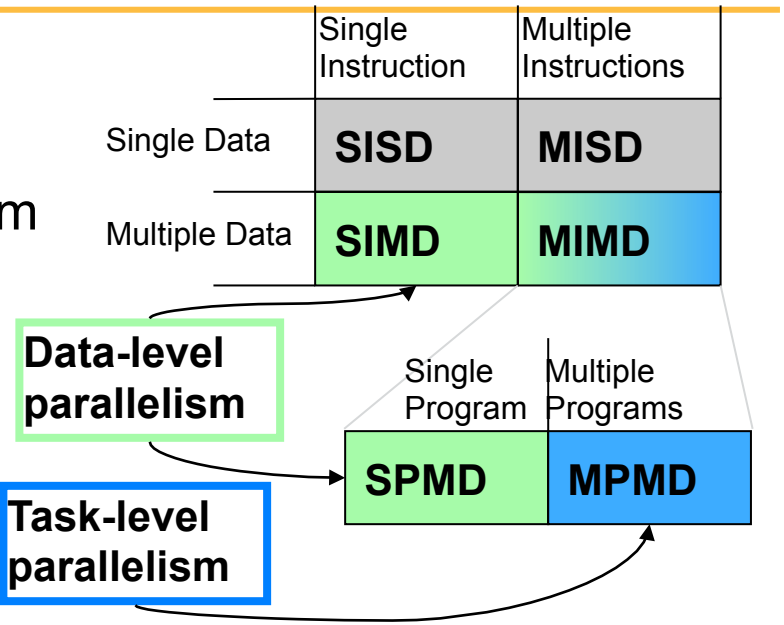
Visualization/debug

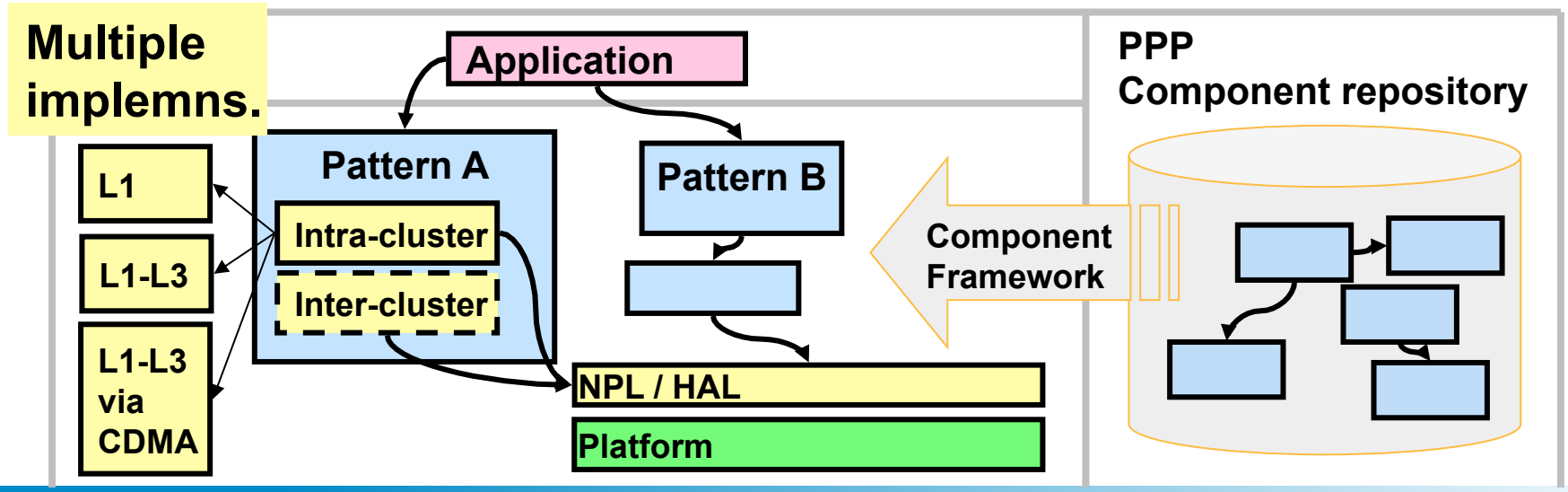


STMicroelectronics

Parallel Programming Patterns

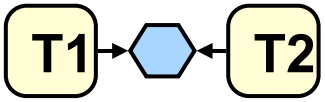
- High-level set of patterns used to parallelize applications
  - Exploiting different types of parallelism
  - Interchangeable pattern implementations
- Component-based (MIND framework)
  - Explicit application capture



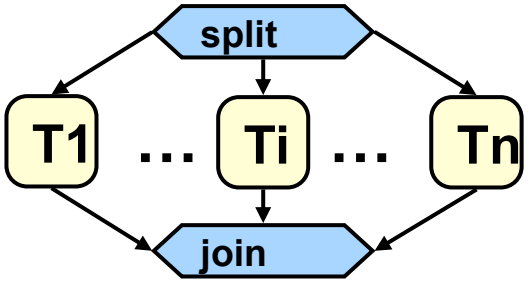


Communication patterns (examples)

- **Exchanger:** `buf = exchange(buf)`
  - Swapping buffers between two participants



- **Queue Iterator:** `nextBuf=writeNext(buf);` `nextBuf=readNext()`
  - Iteration based communication between producer(s) and consumer(s)
    - Single queue
    - Split / Join / Broadcast



- **Synchronized buffer:** `request/release{read,write}(buf)`
  - Synchronized read/write on shared buffer
    - Paired Synchronized buffer:
      Specialization when there are only two participants
    - Segmented Synchronized buffer:
      Specialization to access a large buffer in smaller slices

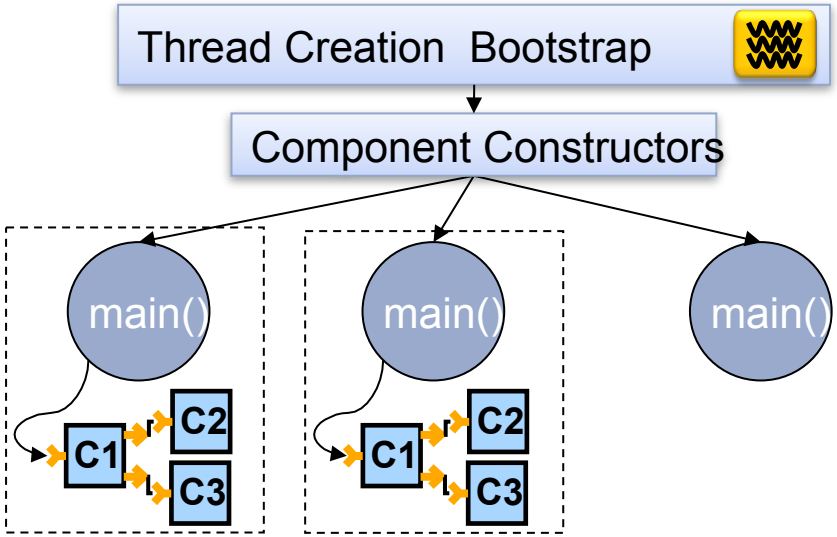
- **FIFO:** `push(); pop()`
  - Packet based streaming between producer and consumer
    - Buffer copy or buffer pointer passing
    - Single queue
    - Split / Join / Broadcast

Memory access patterns

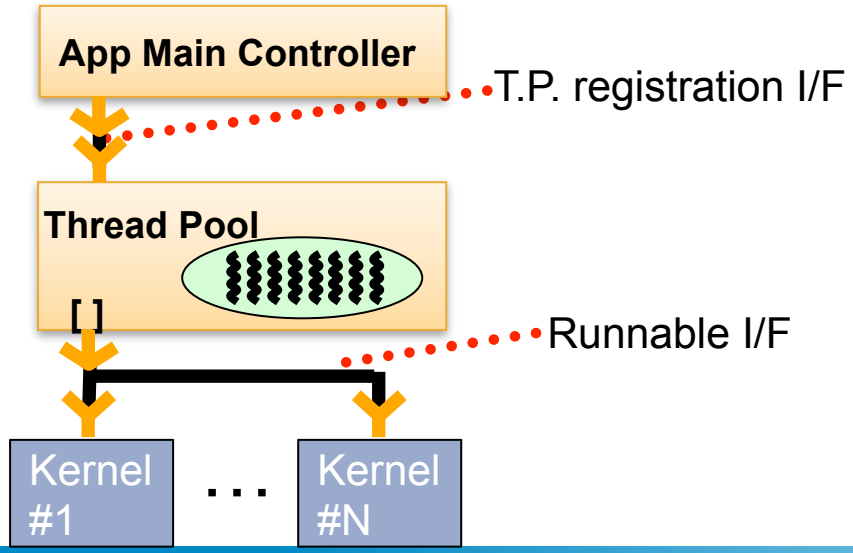
- **Prefetcher:** `key=load(ptrL3,size), ptrL1=get(key), free(key)`
  - Prefetching data from L3 to L1
    - load() programs the DMA prefetch and returns a key without waiting for the transfer
    - get() returns a pointer to L1, blocks until the DMA transfer is completed
    - Also supports 2D arrays: `key=load(ptr, width, height)`

- **Async. Prefetcher:** `load(ptr, size, handler(key)), free(key)`
  - Enables efficient thread-pool execution engines
    - load() programs the prefetch and returns waiting for the transfer
    - handler(key) is invoked by the execution engine when the transfer is completed
    - Also supports 2D arrays: `load(ptr, width, height, handler(key))`

Execution Patterns: thread-based

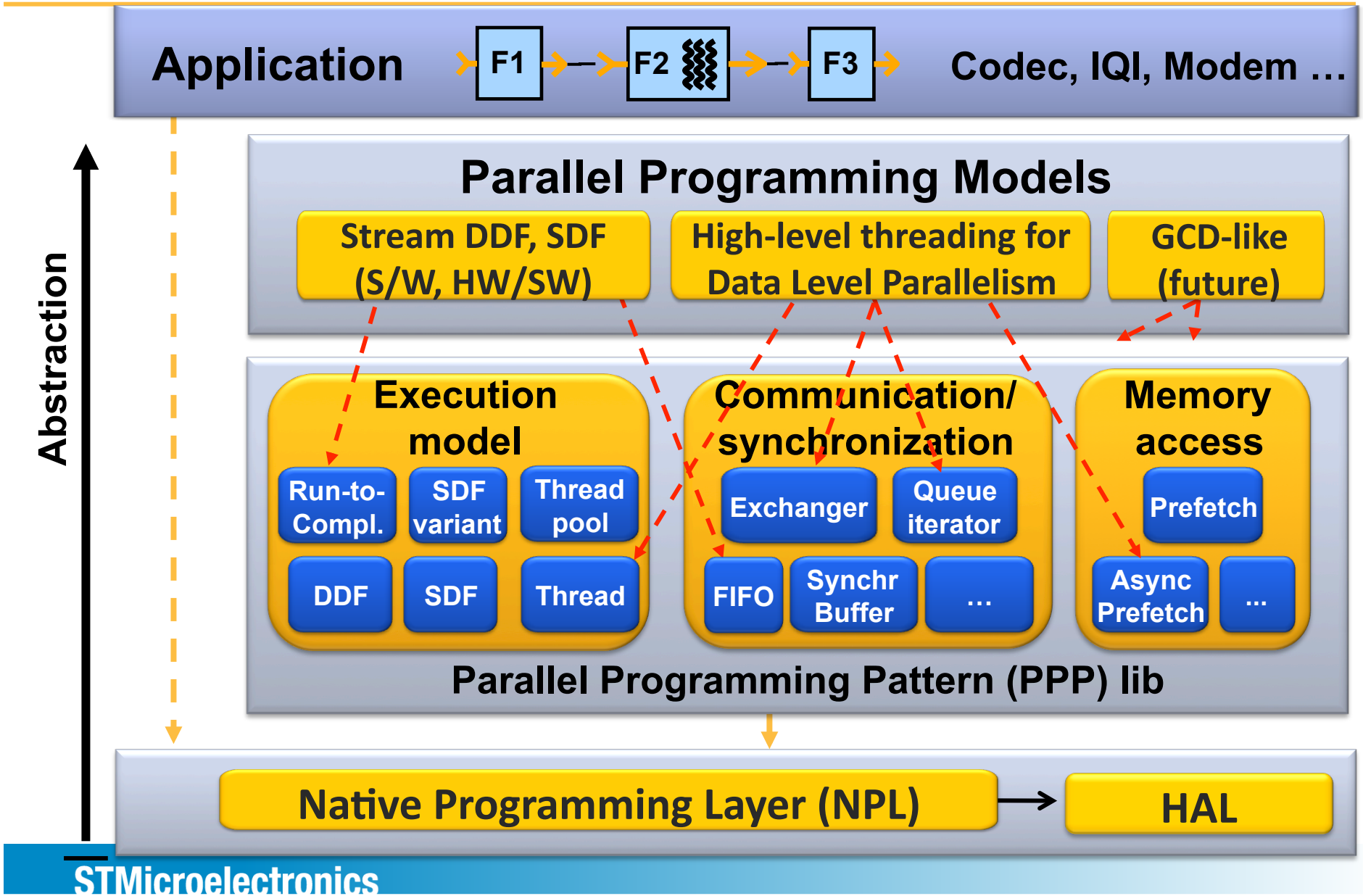


- Static mapping of kernels on a set of platform resources
- Minimal runtime overhead
  - No kernel multiplexing required
- Manual load balancing
  - Similar computational requirements for each kernels



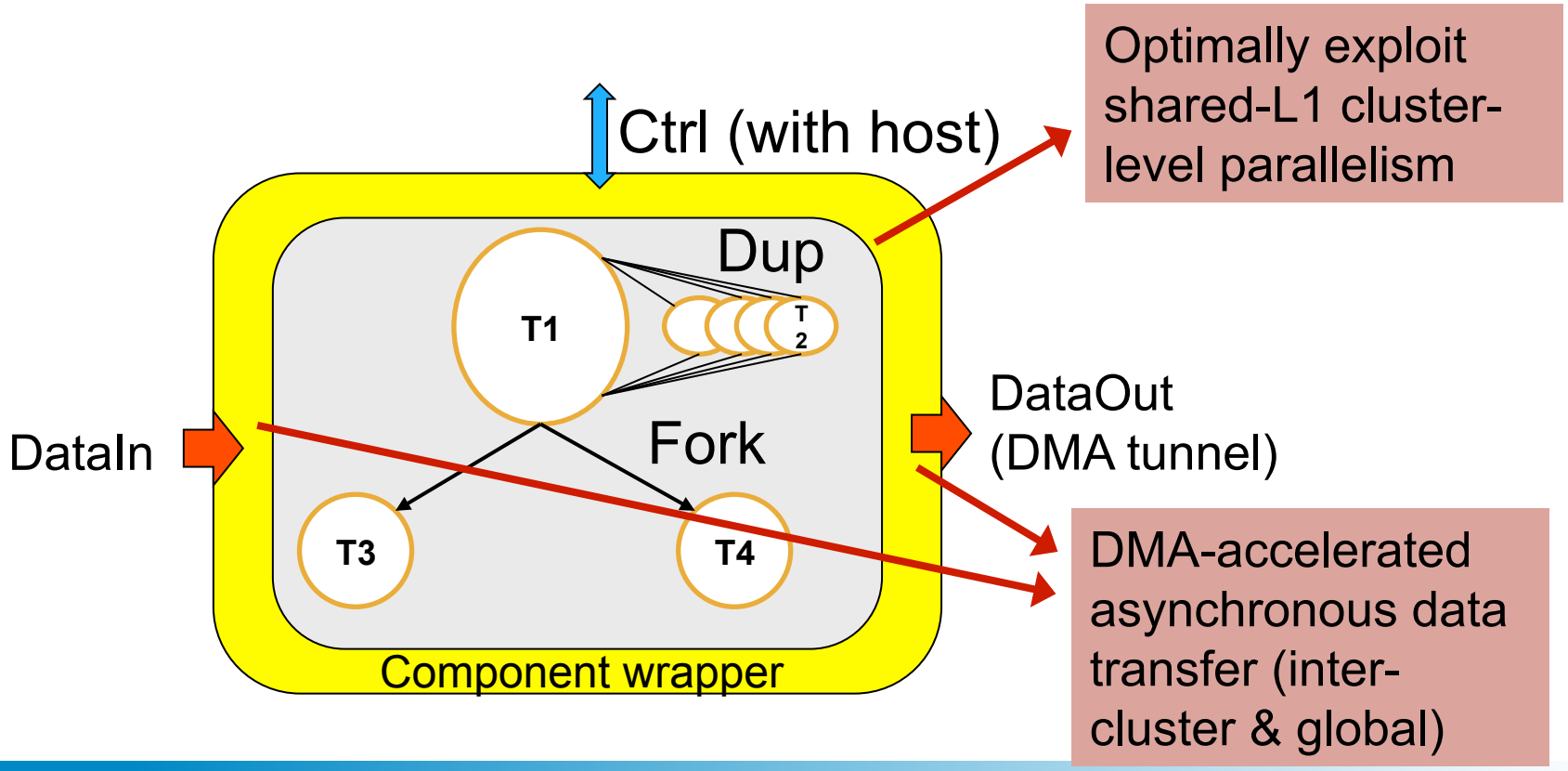
- Dynamic dispatch of kernels on a set of platform resources
  - Some runtime overhead
  - Mux K kernels on R resources
- Dynamic load balancing
  - Different heuristics can be offered
    - Job stealing
    - Affinity

Parallel Programming Models



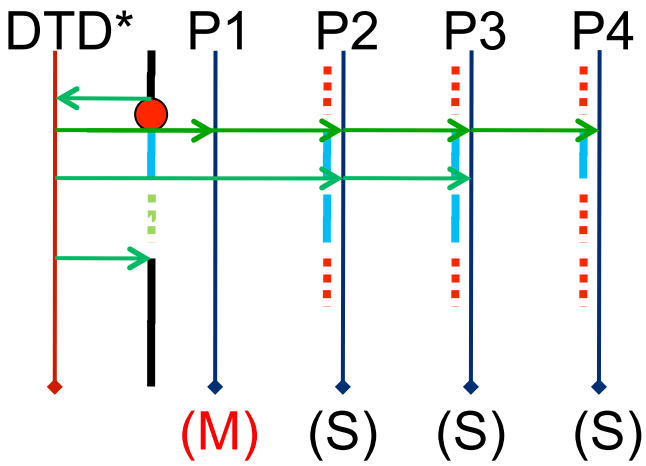
Flexibility is nice, but…

- Programmers need programming model stability
  - Minimize learning curve, build-up SW legacy
- Efficiency is key
  - Highly optimized patterns (HW-accelerated)



Ultra-fast Fork/Dup

```
Fork_join(N) {Body_DOALL;}
```



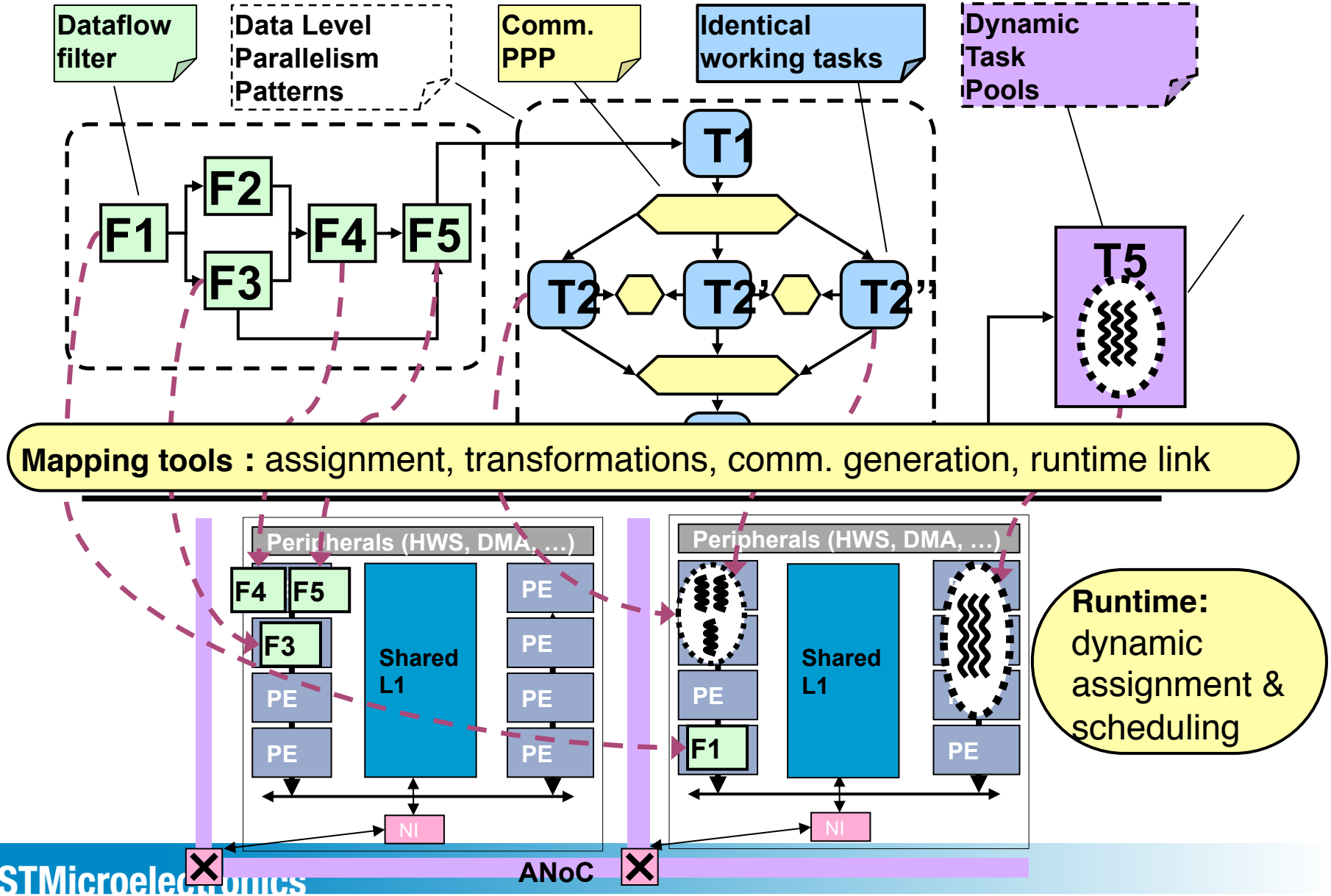
N=6

- Active Polling
- In IDLE
- Working on a “parallel slave thread”
- Working in “master thread”

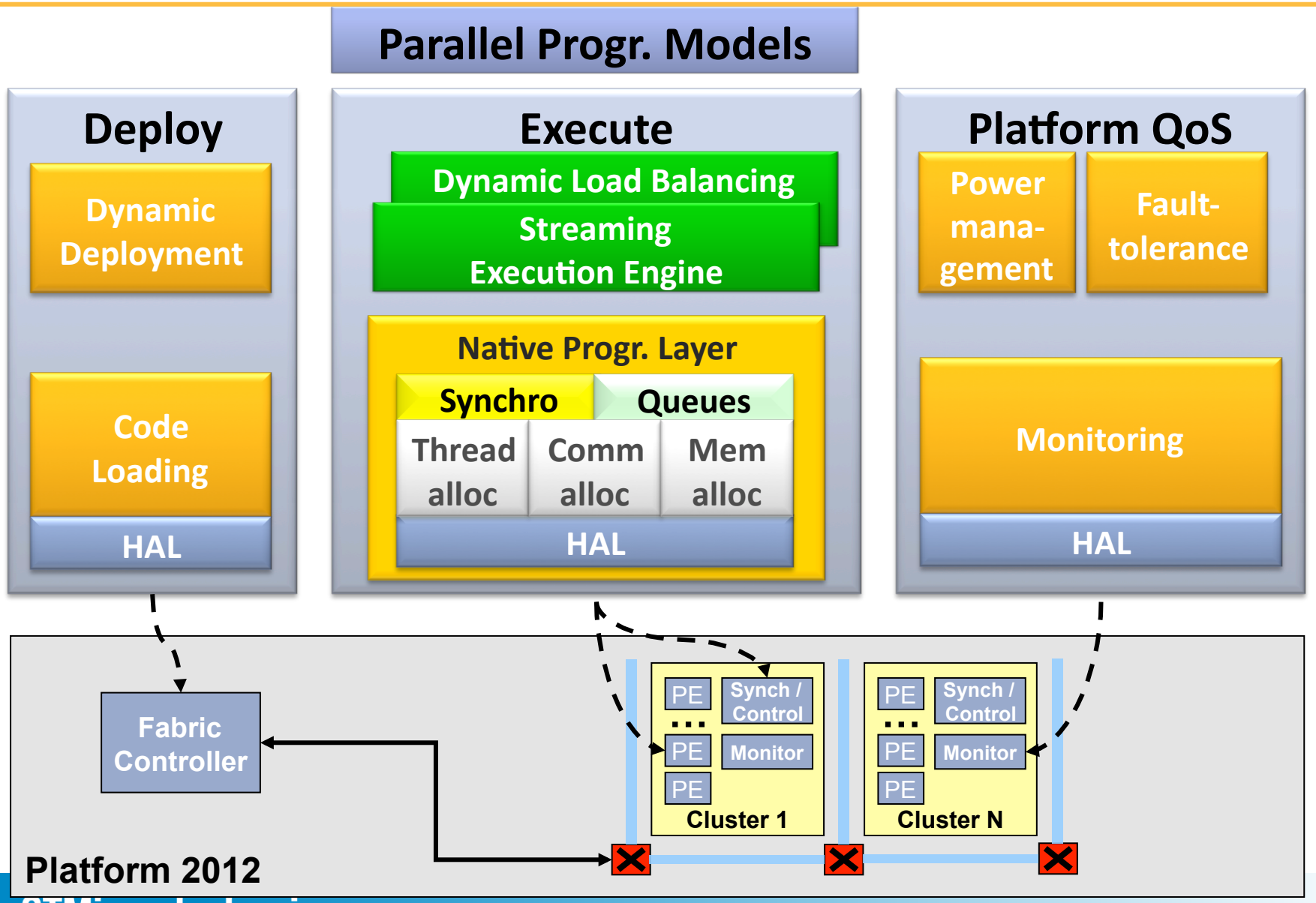
1. The “red dot” represents the call to the DTD API – master is defined when invoking DTD interface
2. Master can execute only work-items of its own fork
3. Barrrier semantics – master active polling until all slaves done
4. If no work-item left, slaves move to IDLE
5. Master continues sequential execution after join

*DTD= *dynamic task dispatcher*
(can be hardware-accelerated)

Application-to-Platform Mapping

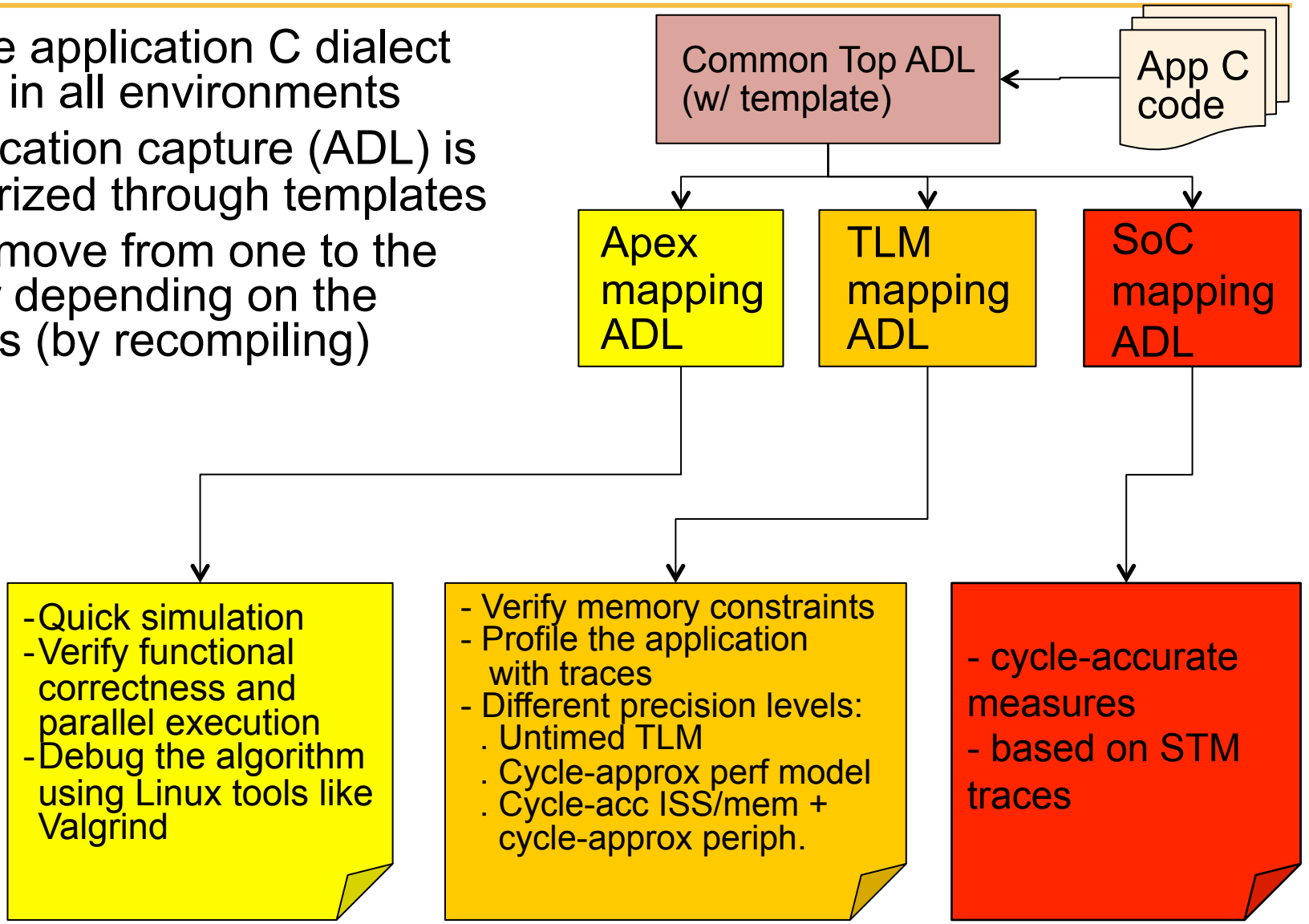


Runtime

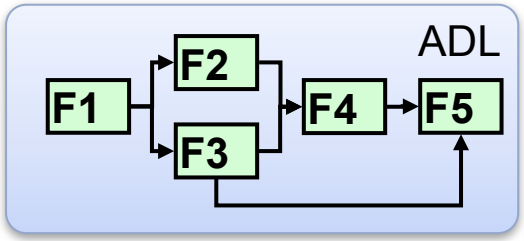

Parallel Progr. Models
Deploy
Dynamic Deployment
Code Loading
HAL
Execute
Dynamic Load Balancing
Streaming Execution Engine
Native Progr. Layer
Synchro
Queues
Thread alloc
Comm alloc
Mem alloc
HAL
Platform QoS
Power mana-gement
Fault-tolerance
Monitoring
HAL
Fabric Controller
PE
Synch / Control
PE
Monitor
PE
Cluster 1
PE
Synch / Control
PE
Monitor
PE
Cluster N
Platform 2012

Application simulation flow

- Same application C dialect used in all environments
- Application capture (ADL) is factorized through templates
- Can move from one to the other depending on the needs (by recompiling)

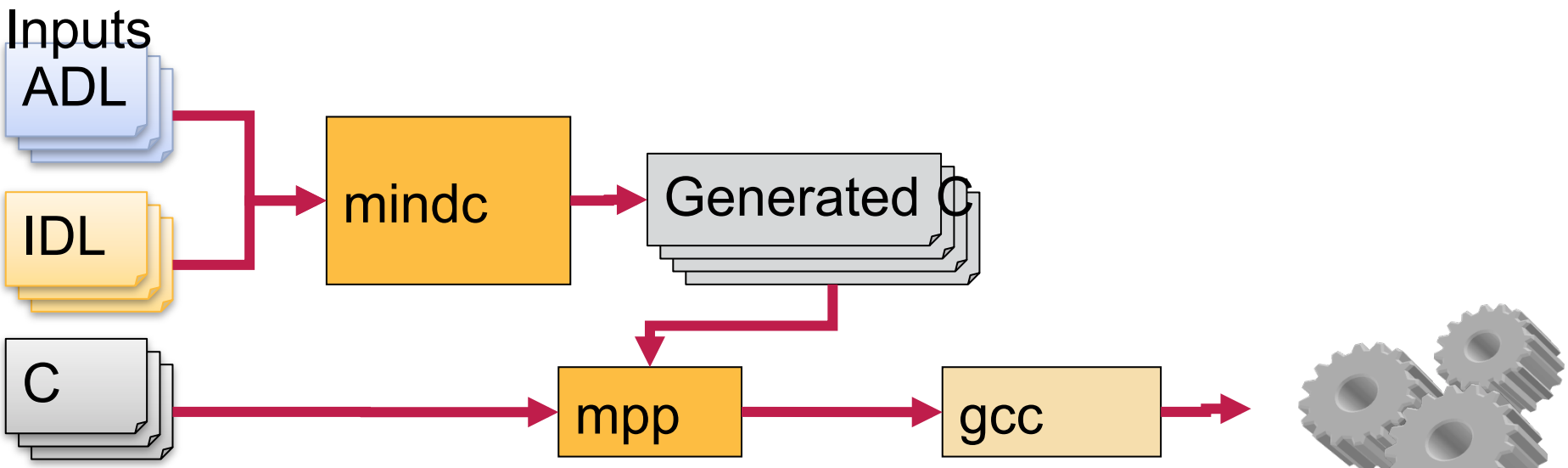


MIND Toolchain




```
interface comm.QueueWriter {
  void createQ(size_t sz,
               int nb);
  void destroyQueue();
  void * readNext();
}
```

```
int METH(itf, func) (void) {
  while(
    (in= CALL(input, readNext)())
      !=0) {
    compute_fct (in,out,PRIVATE.arg);
  }
}
```



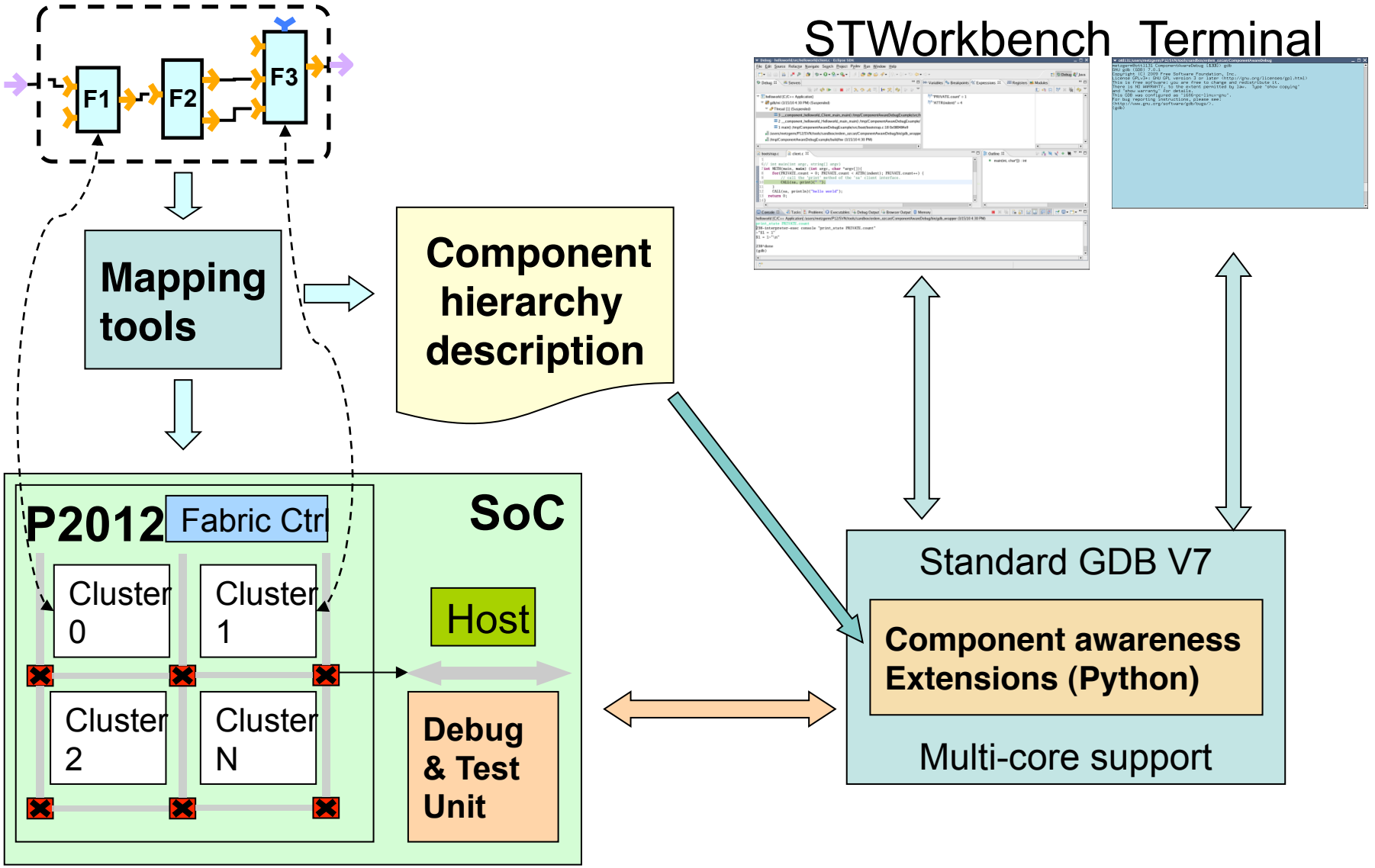
ADL : Architecture Description Language
IDL : Interface Definition Language
mindc : MIND ADL/IDL parser and C code generator
mpp : MIND PreProcessor

Executable/
Loadable binary

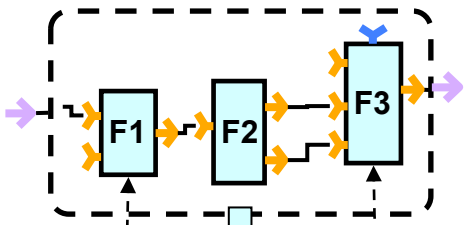
MIND: Component framework

- Implementation of the *Fractal* component model
  - http://fractal.ow2.org/
- Defines 3 languages/dialects
  - ADL describe the architecture of the application as a set of components bound to each-others
  - IDL defines the type of component interfaces
  - C dialect defines a set of macro to capture “object-oriented” notions
- Provides
  - **`mindc`** : a compiler that generates C “glue-code” from ADL and IDL
  - **`mindEd`** : IDE based on Eclipse (3.6) that provides, among others, graphical editor for ADL files
- Open source (LGPL) available on OW2 consortium
  - http://mind.ow2.org/

Debug flow


F1
F2
F3
Mapping tools
Component hierarchy description
STWorkbench
Terminal
P2012
Fabric Ctrl
SoC
Cluster 0
Cluster 1
Cluster 2
Cluster N
Host
Debug & Test Unit
Standard GDB V7
Component awareness Extensions (Python)
Multi-core support

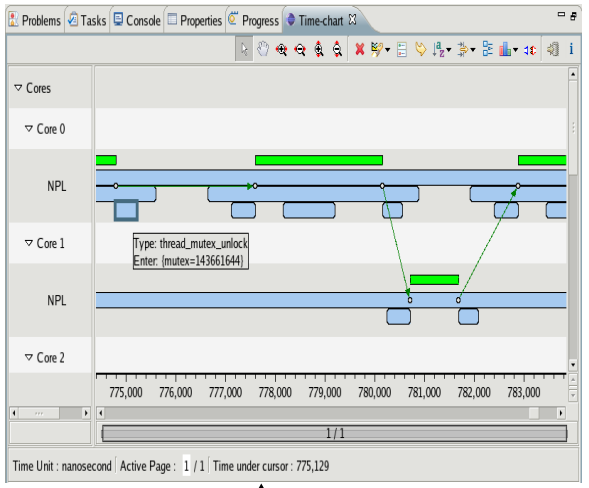
Visualization & Analysis flow

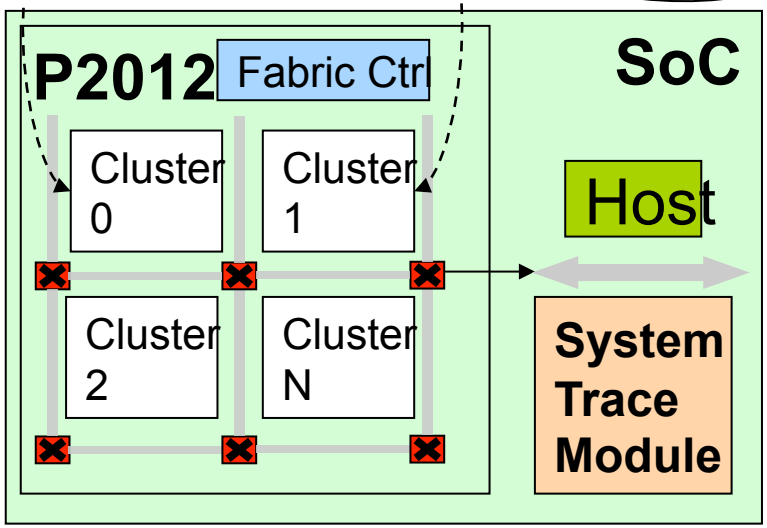


Mapping tools

- Correspondence between application description and platform resources
  - Application mapping results
  - Used resources
  - Symbolic names / physical IDs

Visualization and analysis tools (in STWorkbench)





Low-level Traces

Trace Database

OpenCL : Standard of the convergence

Handles dynamism of adaptive applications

OpenCL

Handles memory constrained systems

Scalability and "load balancing"

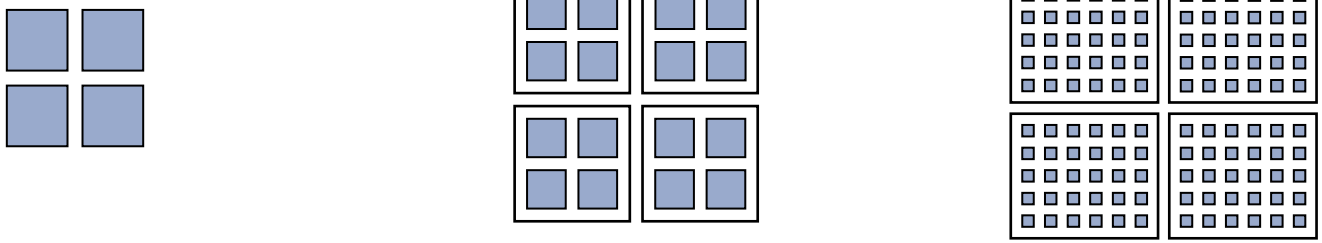
Provides access to all the system parallelism
Task, SPMD/MIMD, SIMD

ARM AMD intel IBM Imagination NVIDIA ATI ARM QUALCOMM

More parallelism

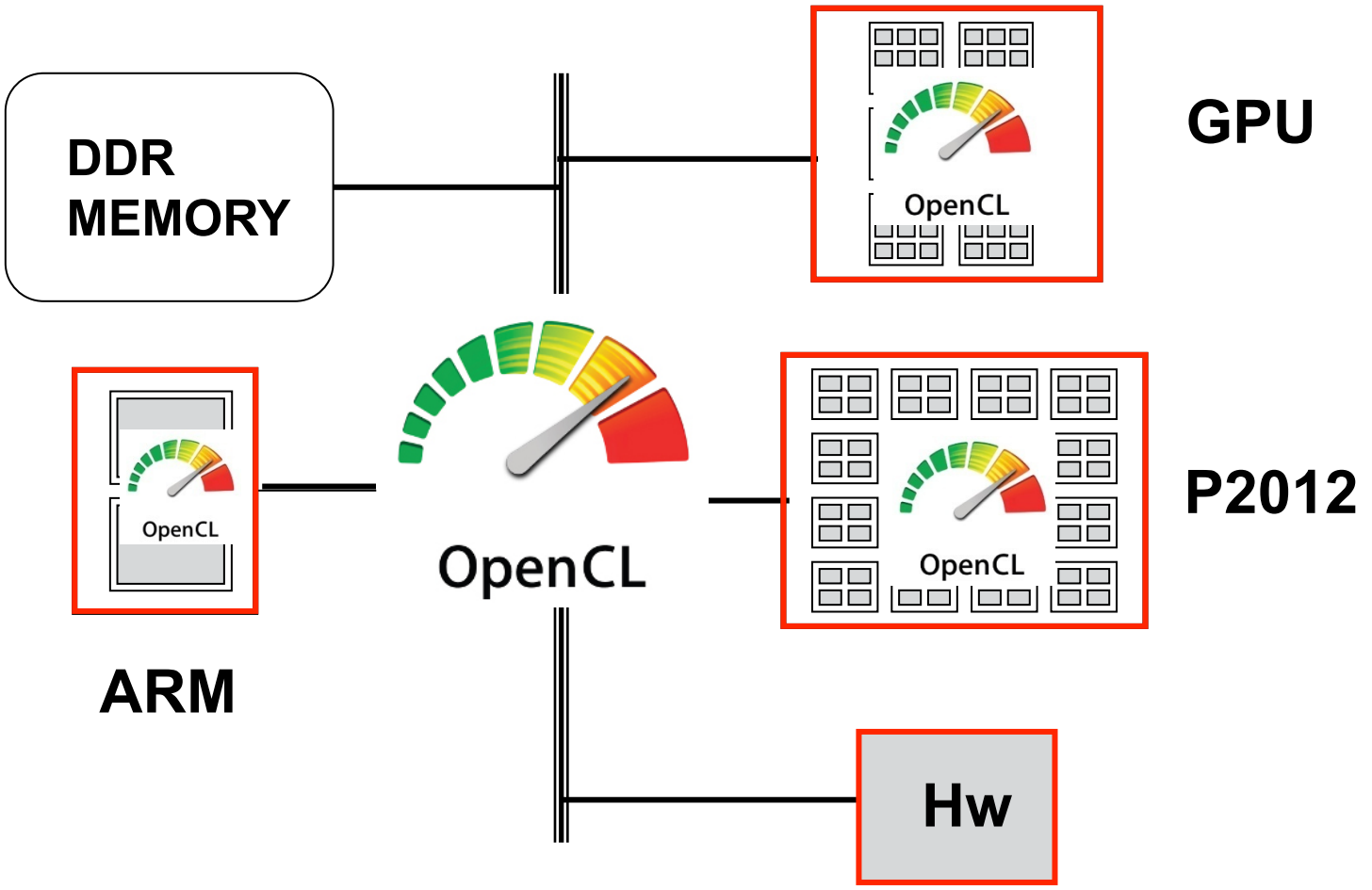
More programmability

Multi-core CPU | Many-core | GPU

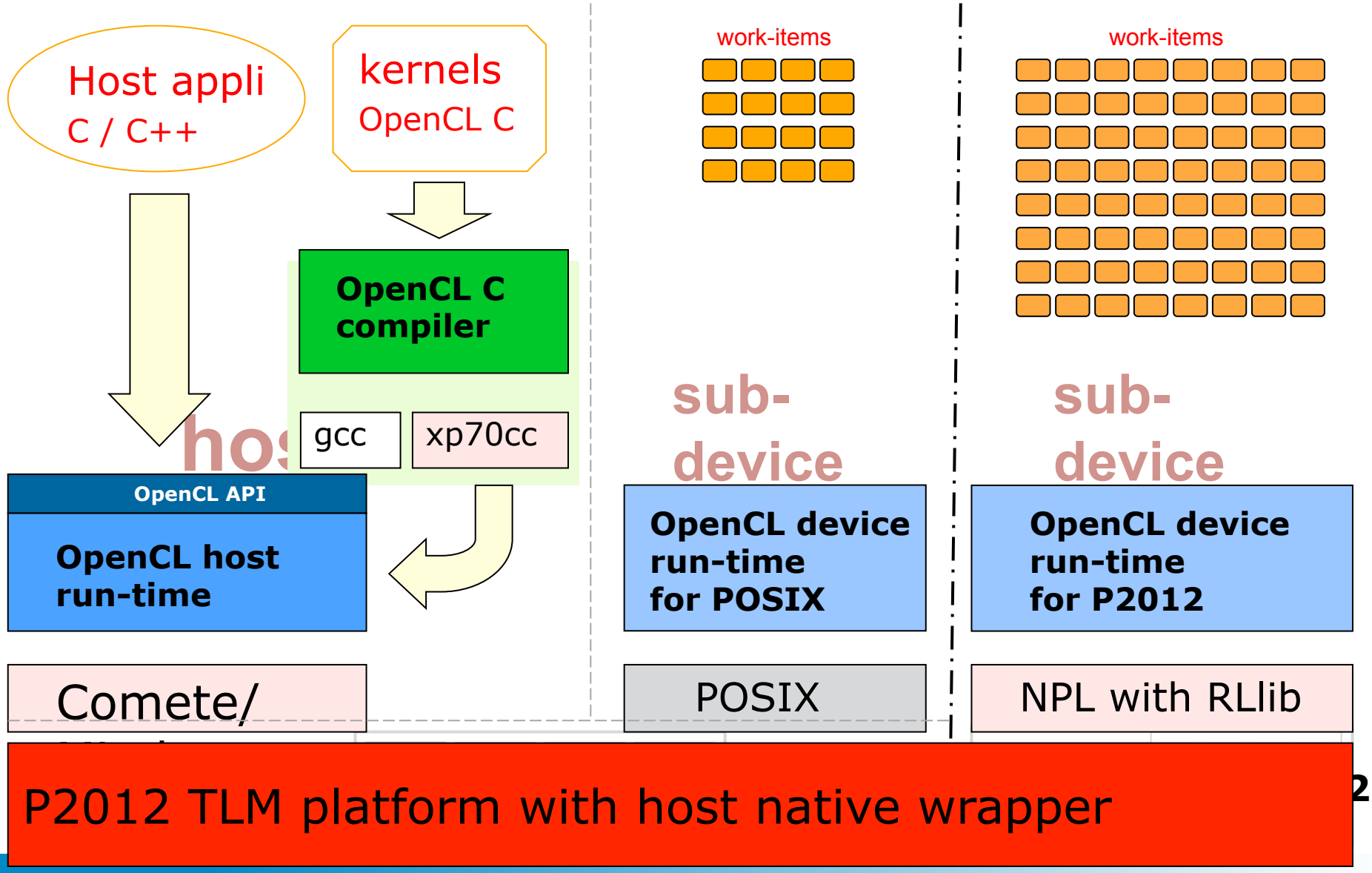


Tapping into a large and growing SW ecosytem

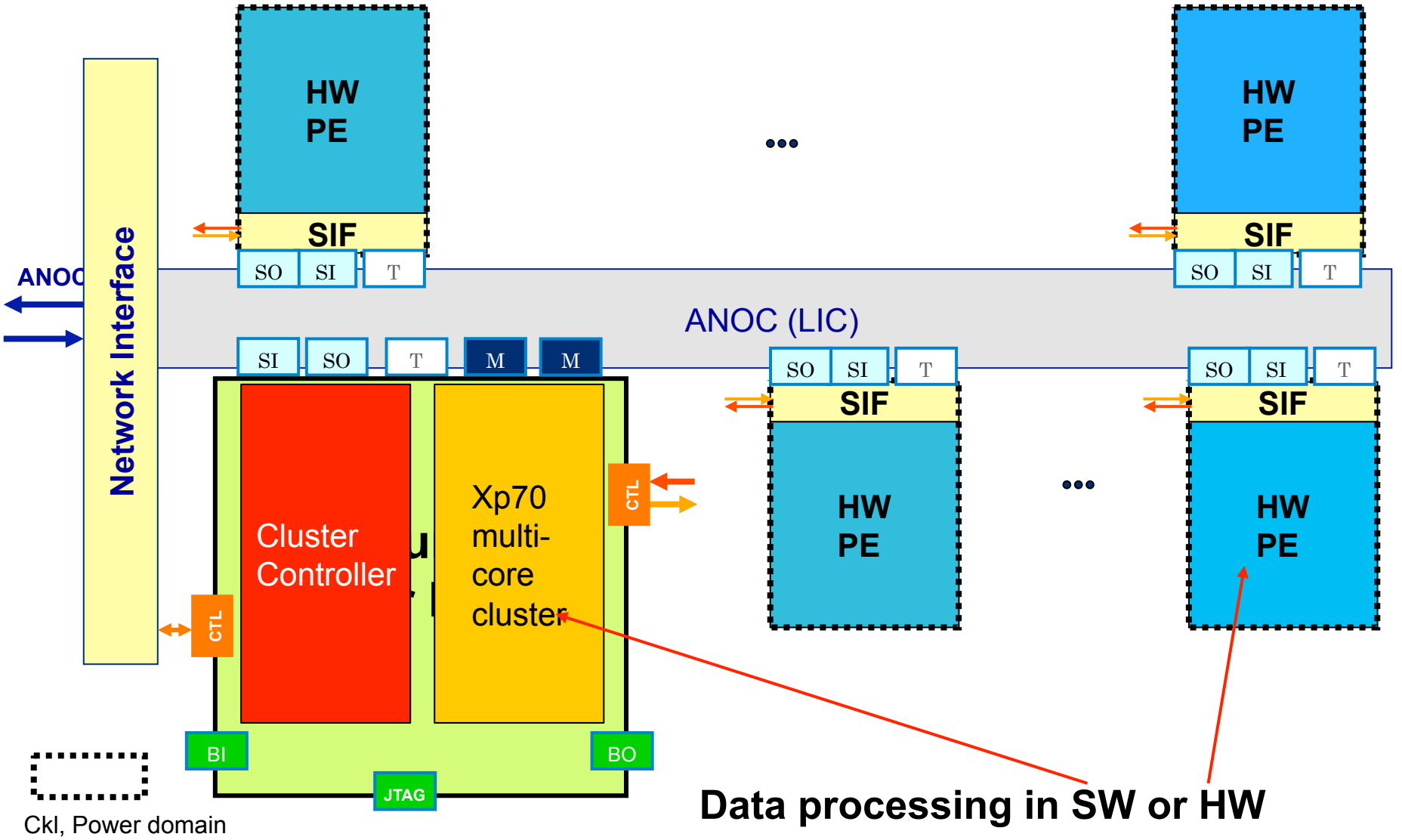
SoCs and OpenCL



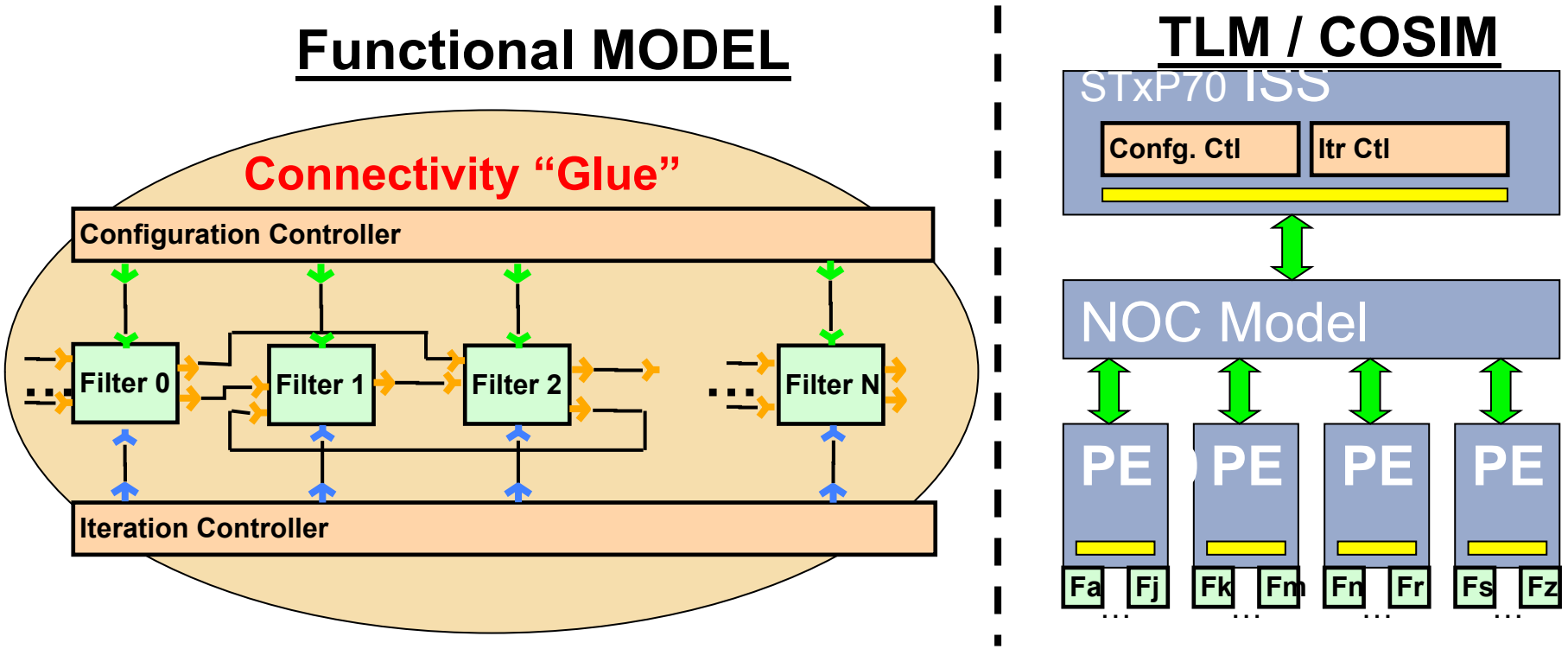
CLAM architecture



What about HW-acceleration?

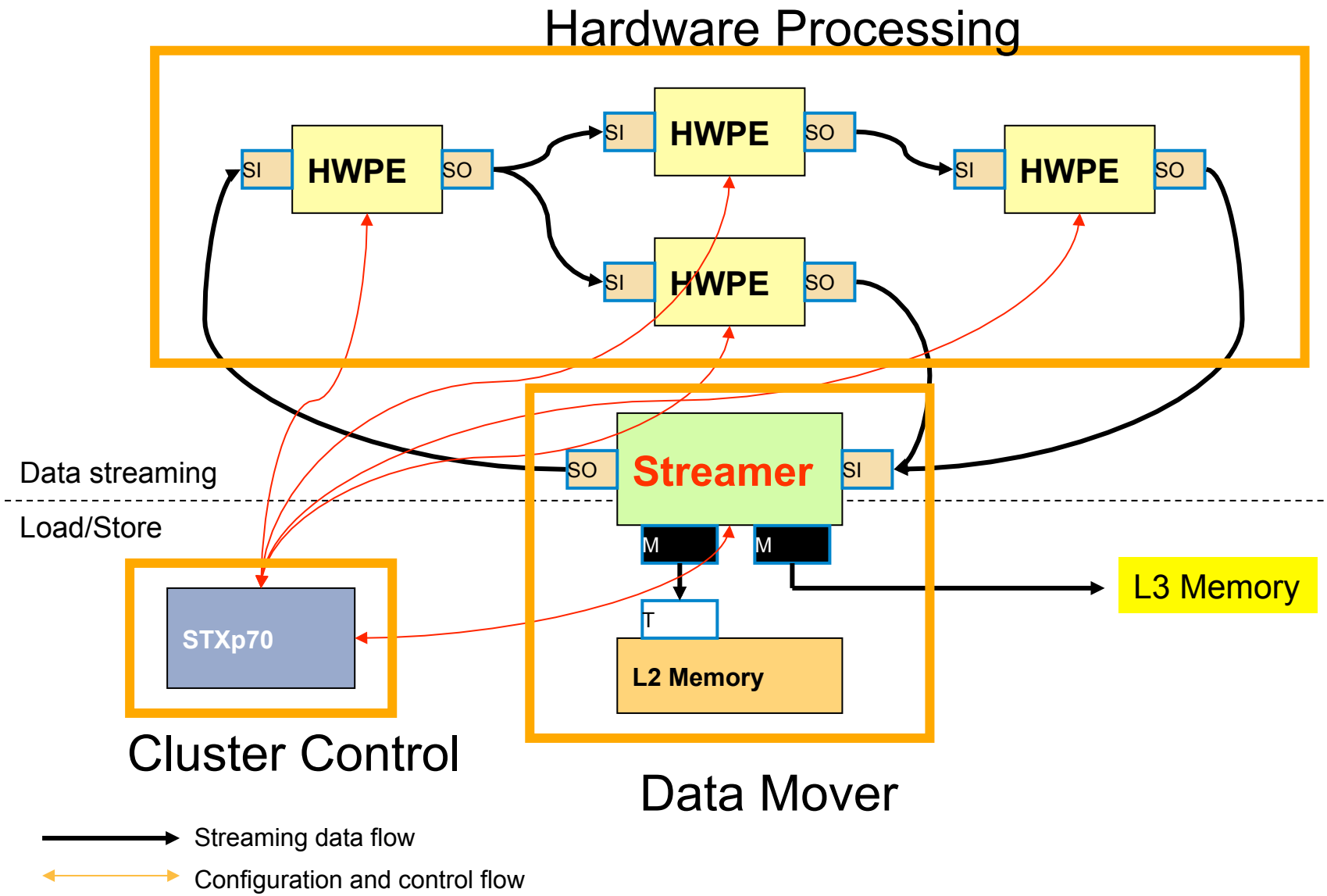


Mapping to HW/SW Platform

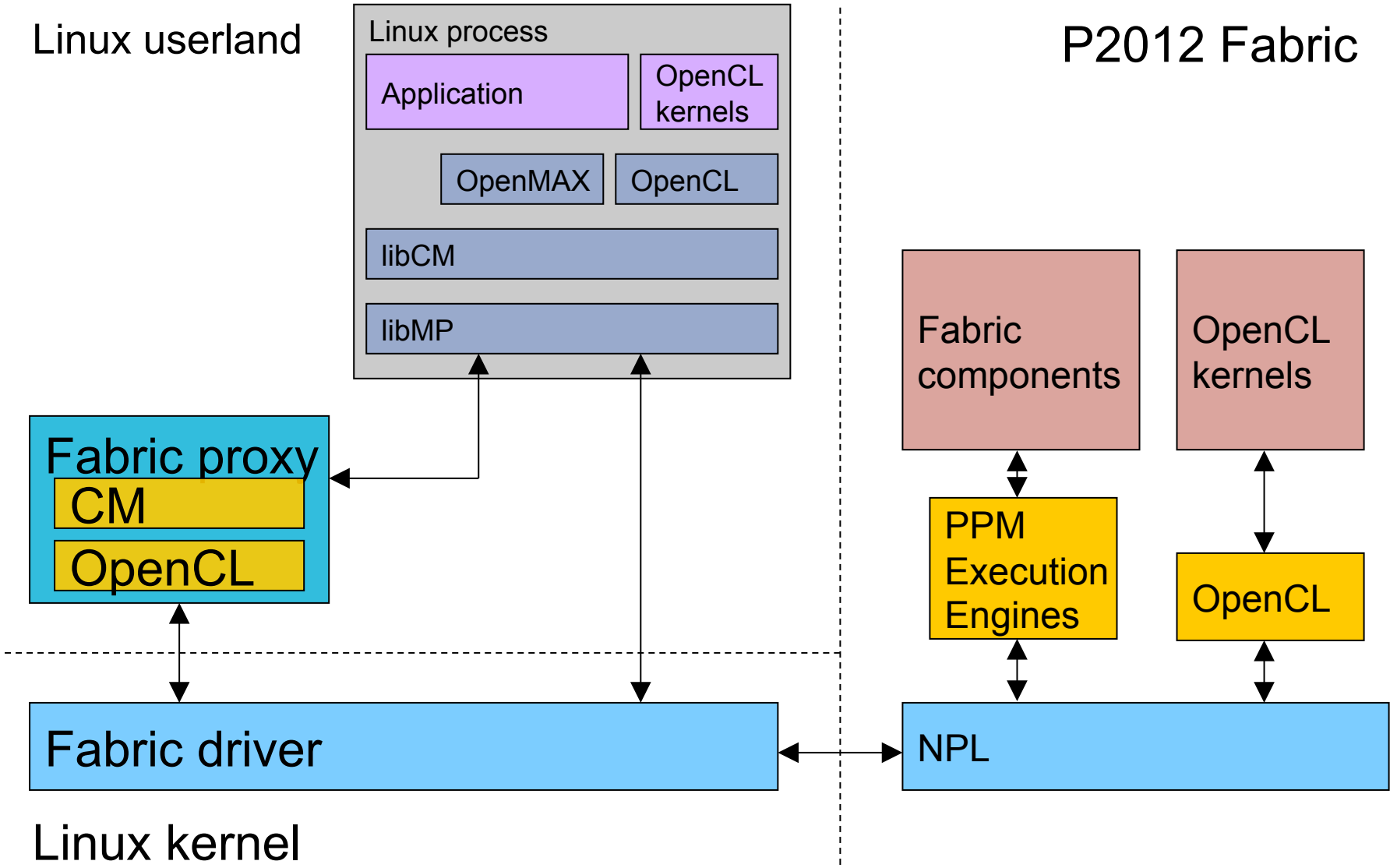


- Application capture using DF variant prog. model
- Host execution (APEX) for functional validation
- Automatic control code generation for TLM/COSIM for perfomance analysis

HW-SW Interaction in P2012



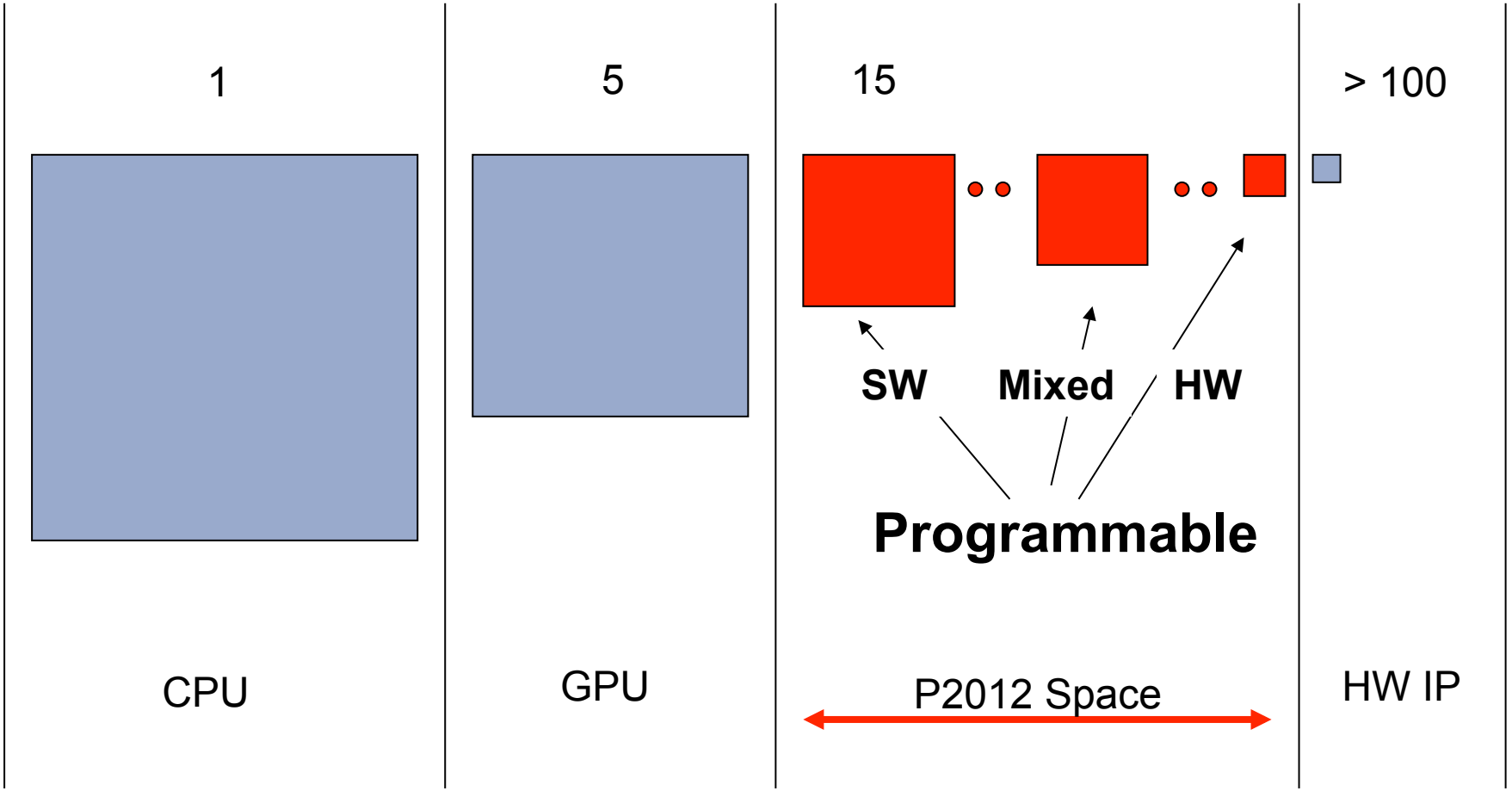
SoC integration: SW stack



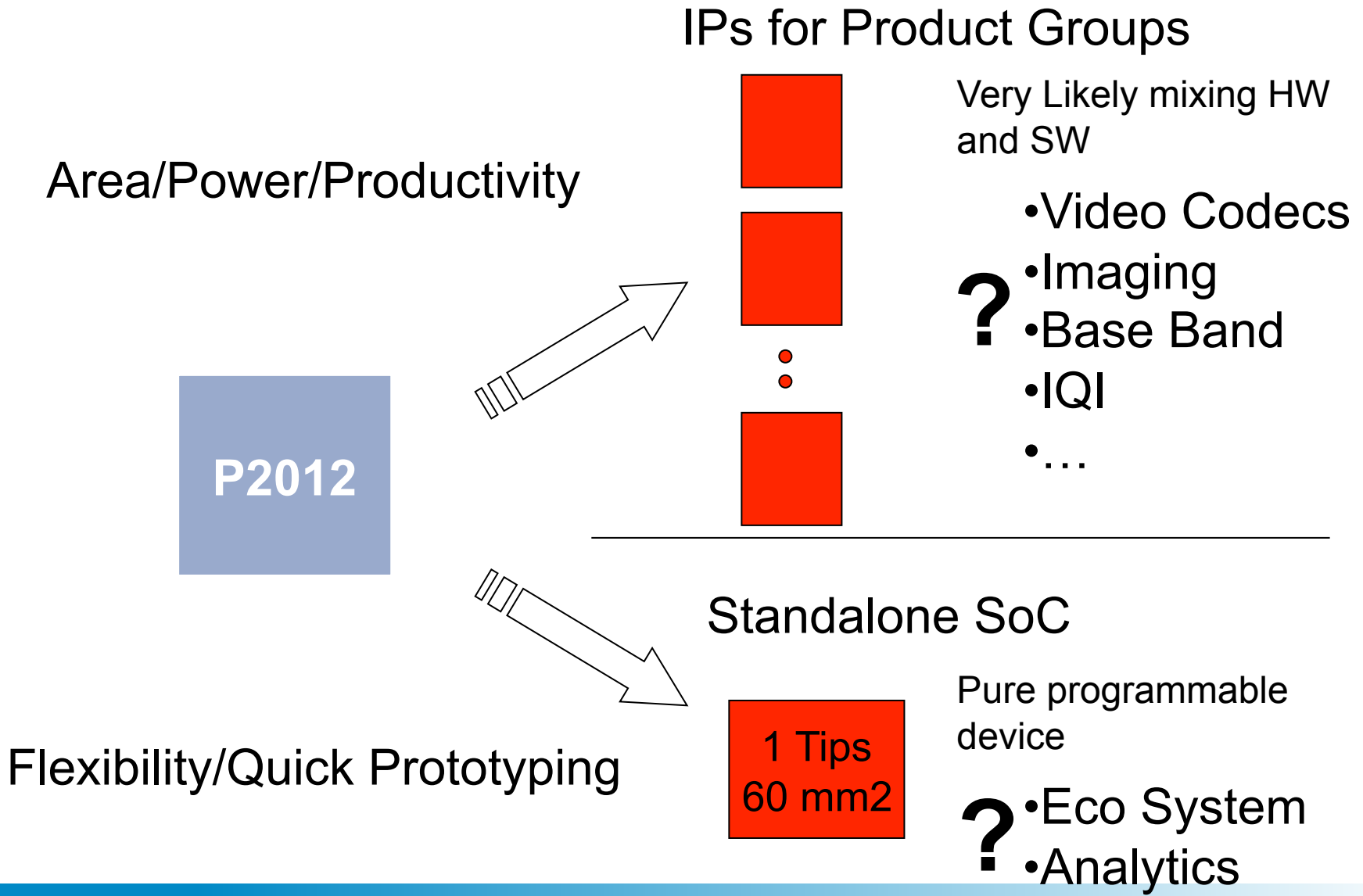
Final thoughts

Summing up: P2012 Space

Gop/s/mm2 in 32 nm

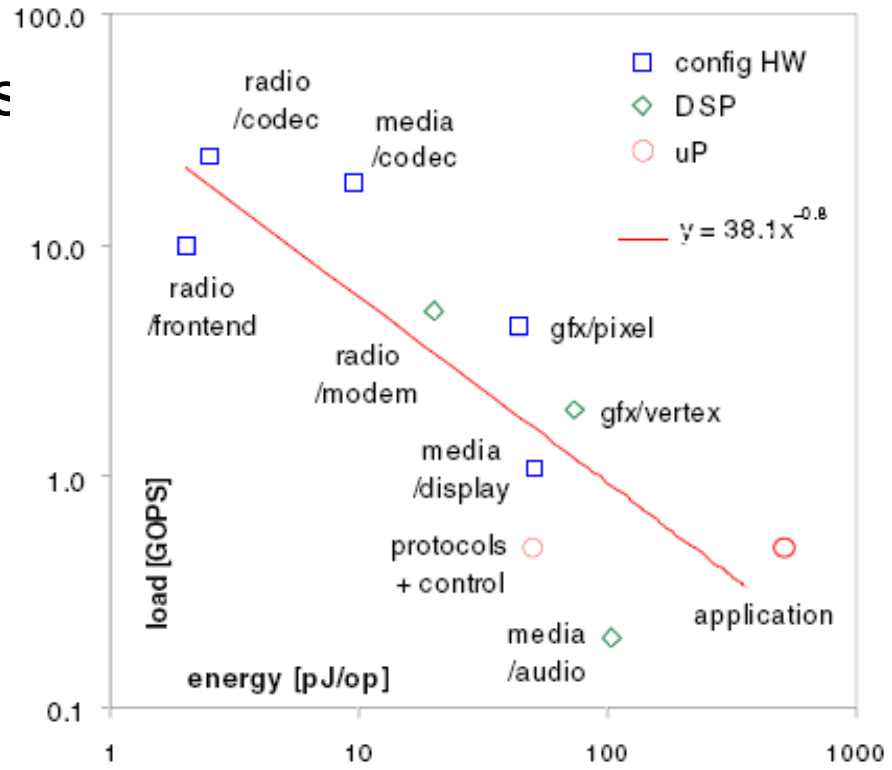


And to do what ?



Conclusion

- Homogeneous processor fabrics are conceptually appealing, but most likely not an industry-viable answer
  - heterogeneous IOs/L3
  - heterogeneous applications
  - …**and** cost !!!
- GOPS/W & GOPS/mm^2 are a hard reality
- We don't need homogeneity, we need modularity!
- The real challenge is how to design a scalable modular heterogeneous many-core system
- … and how to program it!



[NXP09]

Thank you!