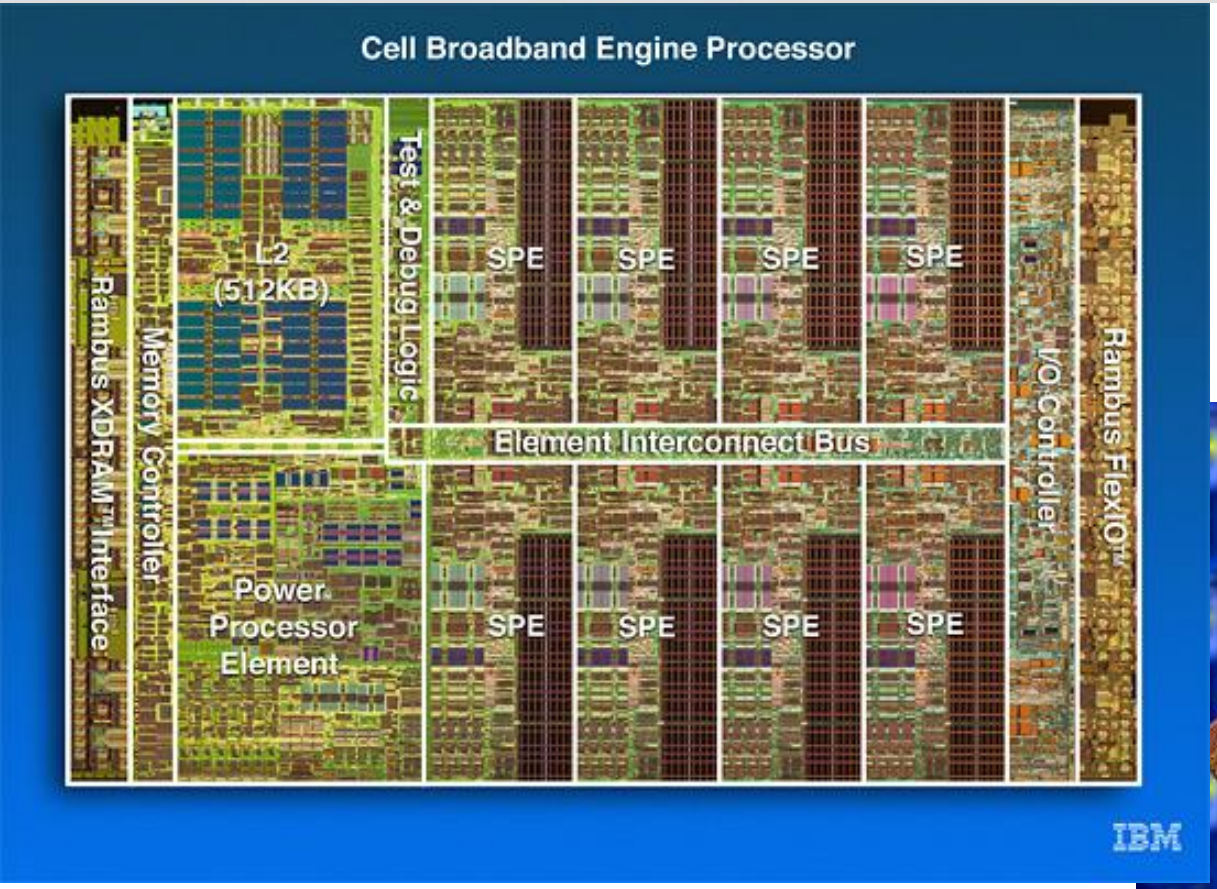
MPSoC Programming In General

Kai Huang

What is MPSoC?

- Multi-Processor System-on-Chip
  - Multiple, usually heterogeneous, processing elements, a memory hierarchy, and I/O components, linked by an on-chip interconnect
- What is it used for ?
  - Meet the performance needs of domain specific applications while limiting the power consumption
- What is MPSoC programming?
  - To develop applications for MPSoC platforms

What is MPSoC?

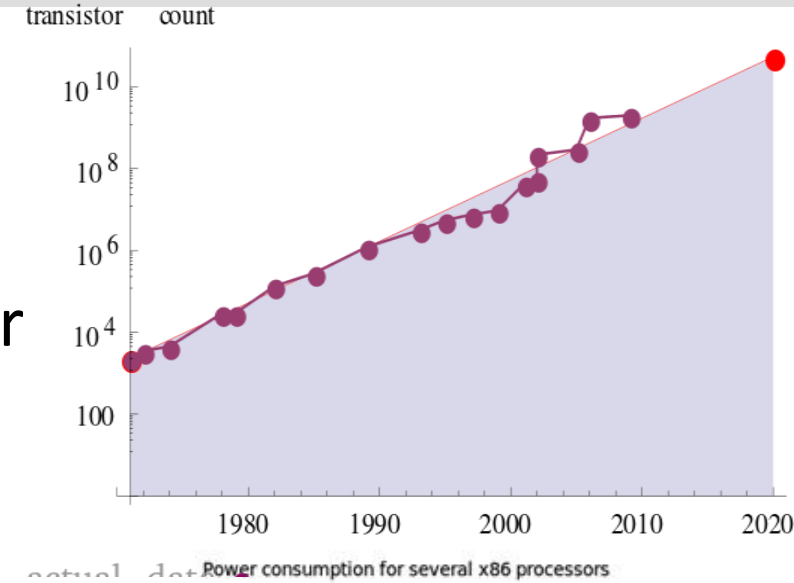


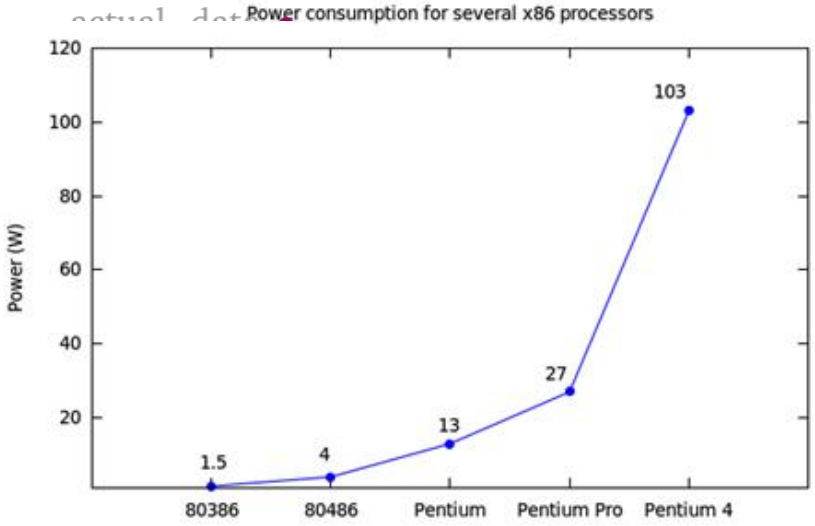
IBM CELL Broadband Engine

Intel SCC

Why is MPSoC?

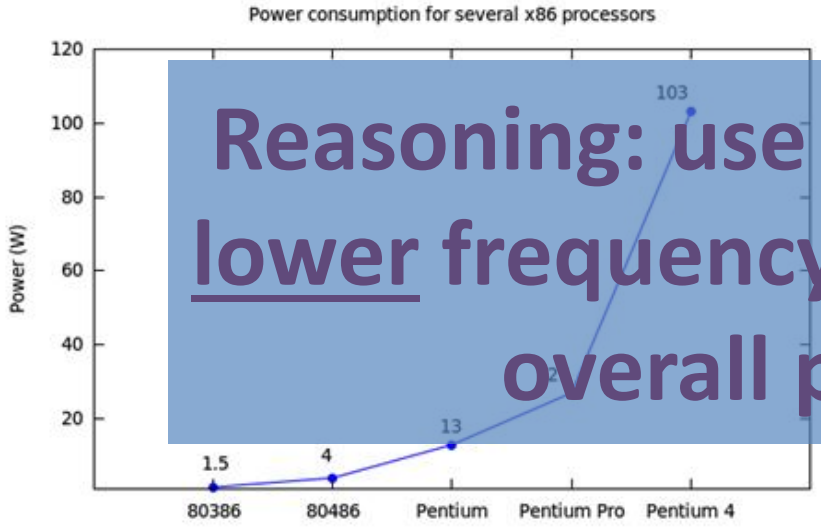
- Moors Law
- Power Wall
  - The Power Wall means faster computers get really hot.

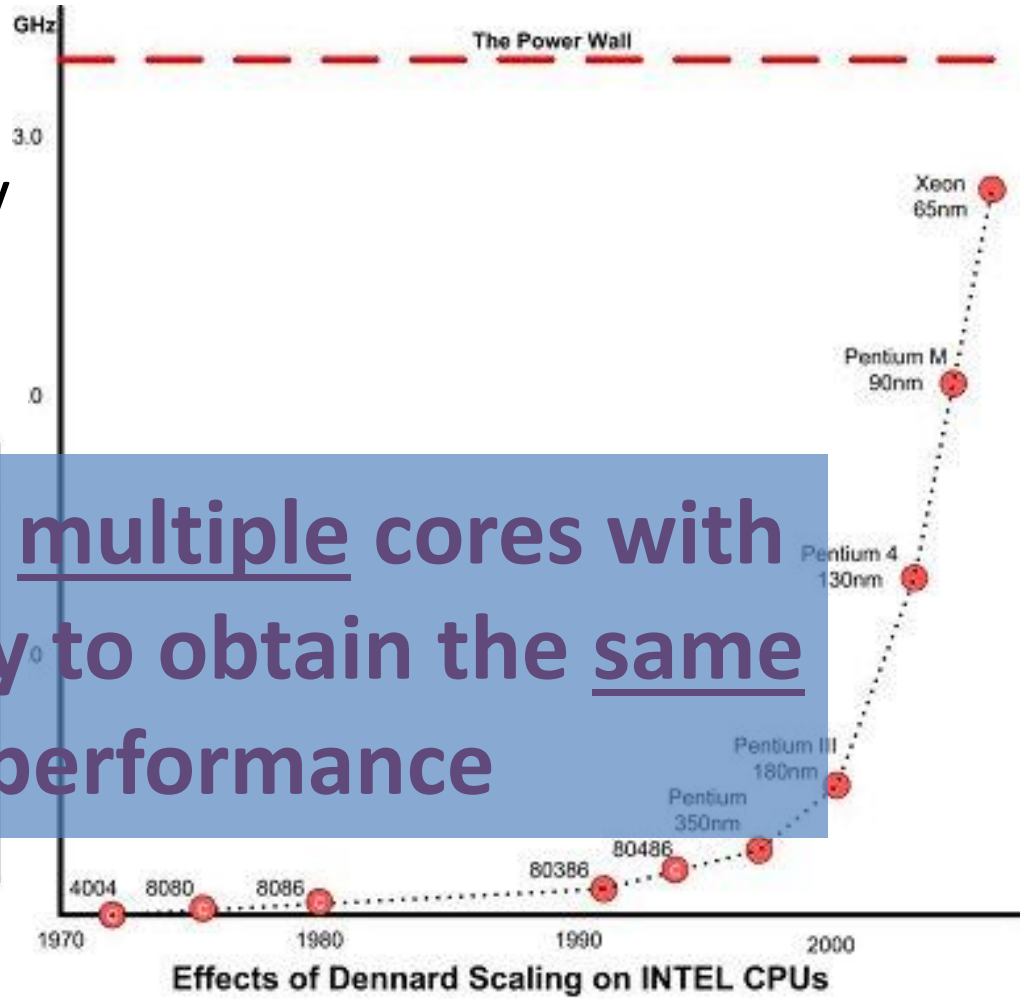




Power Wall

- Law of Physics: All electrical power consumed is eventually radiated as heat





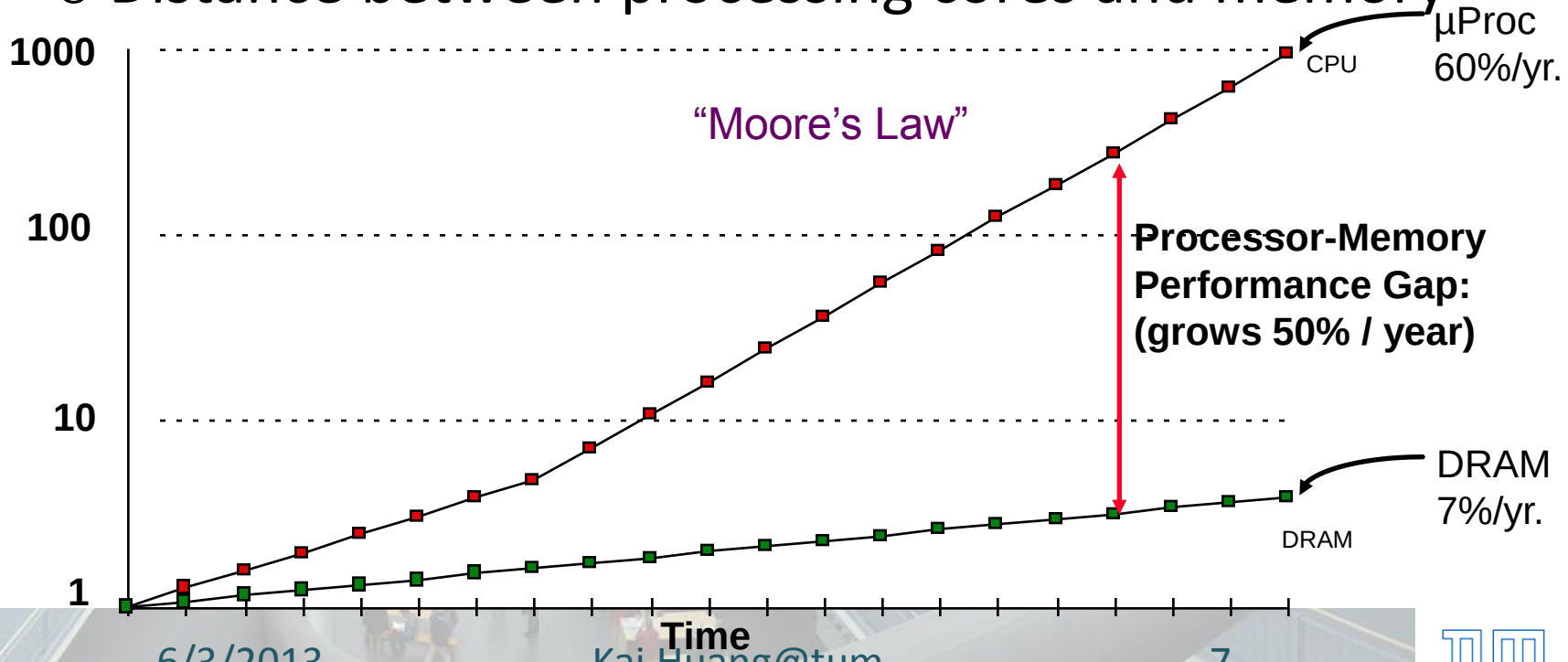
Reasoning: use multiple cores with lower frequency to obtain the same overall performance

What Is the Problem of MPSoC?

- Memory Wall
- ILP Wall
- Power is still a issue

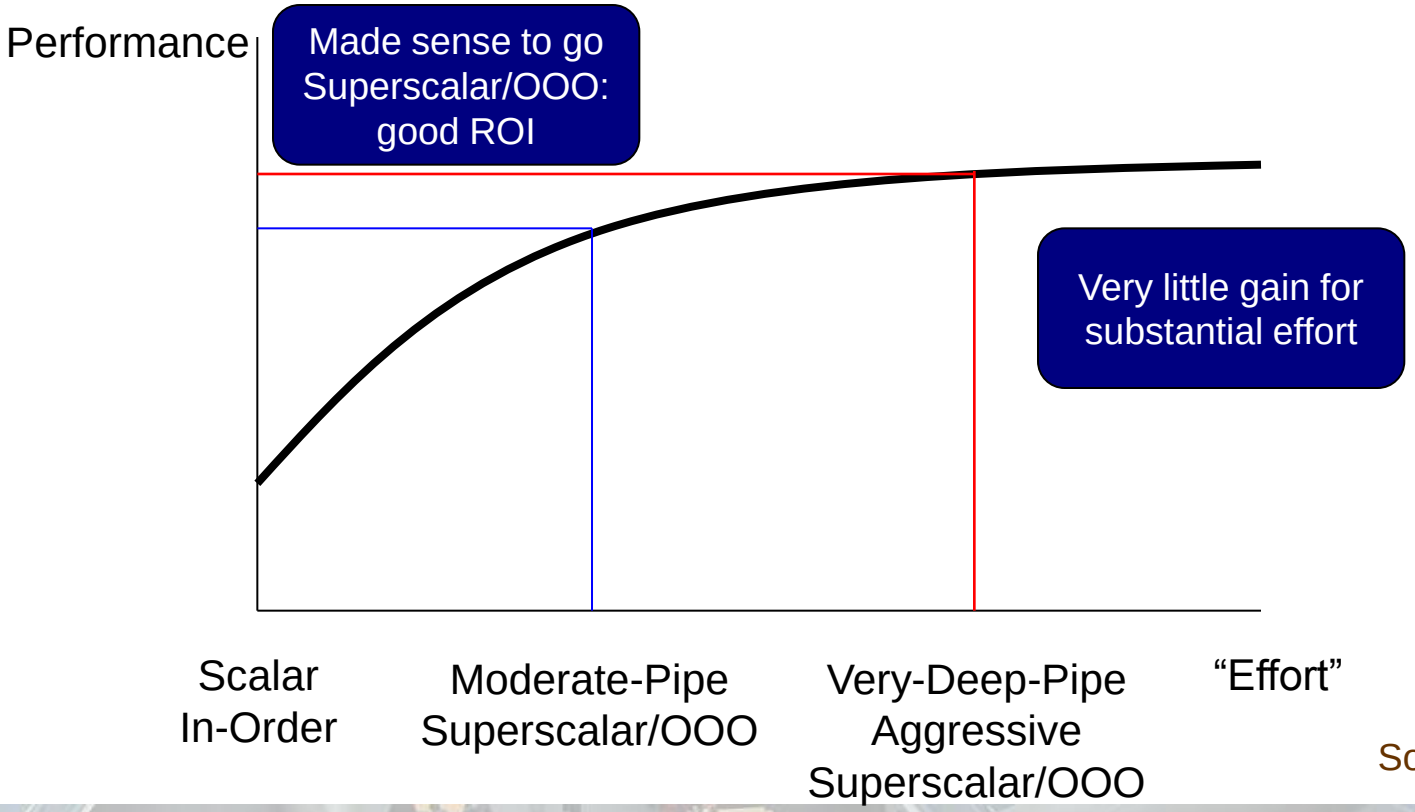
Memory Wall

- The Memory Wall means 1000 pins on a CPU package is way too many
  - Bandwidth limit
  - Distance between processing cores and memory



ILP (Instruction Level Parallelism) Wall

- ILP Wall means a deeper instruction pipeline really means digging a deeper power hole



Source: G. Loh

Power Wall for MPSoC

Reasoning: packing more transistors needs deeper sub micro CMOS techniques which results in larger leakage current

What Is the Problem of MPSoC?

- Memory Wall
- ILP Wall
- Power is still a issue

➢ How to program an MPSoC affects a lot!

Parallel Programming Success Stories

- Scientific/Super computing
  - Where data parallelism is abundant
- Data Centers
  - Independent queries
  - Few shared writes
  - Buying more hardware is cheaper and faster than efficient programming
- Computer Graphics
  - Near infinite, homogeneous parallelism
  - Portable libraries: OpenGL, DirectX

Language Classification

No Memory Concept

Esteral

Temporal logic

Complete Memory Abstraction

Garbage Collection

Python

Java

Haskell

Stack Memory Abstraction

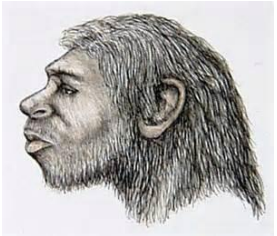
malloc

free

Fortran

C/C++

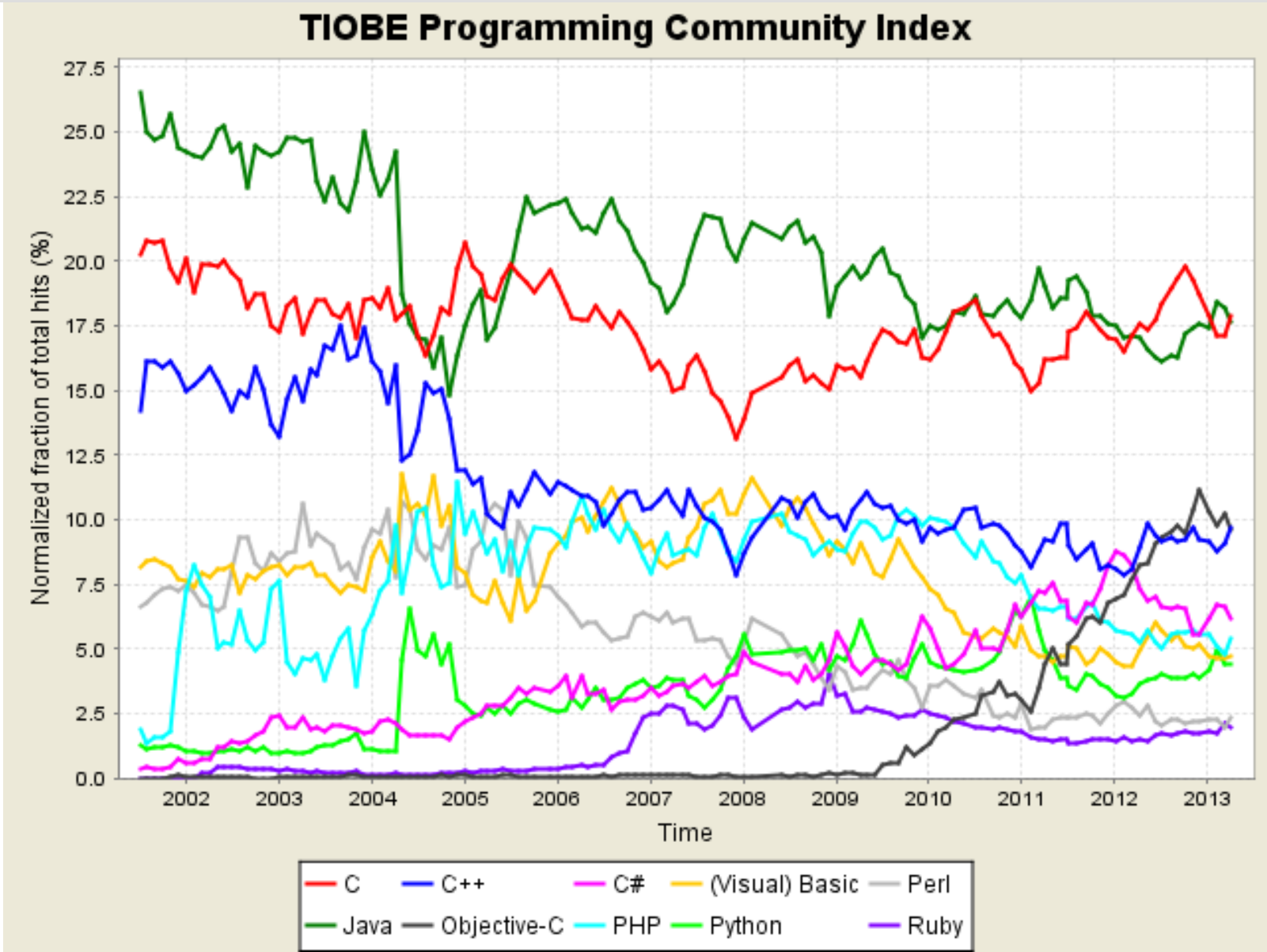
Stack Memory Abstraction



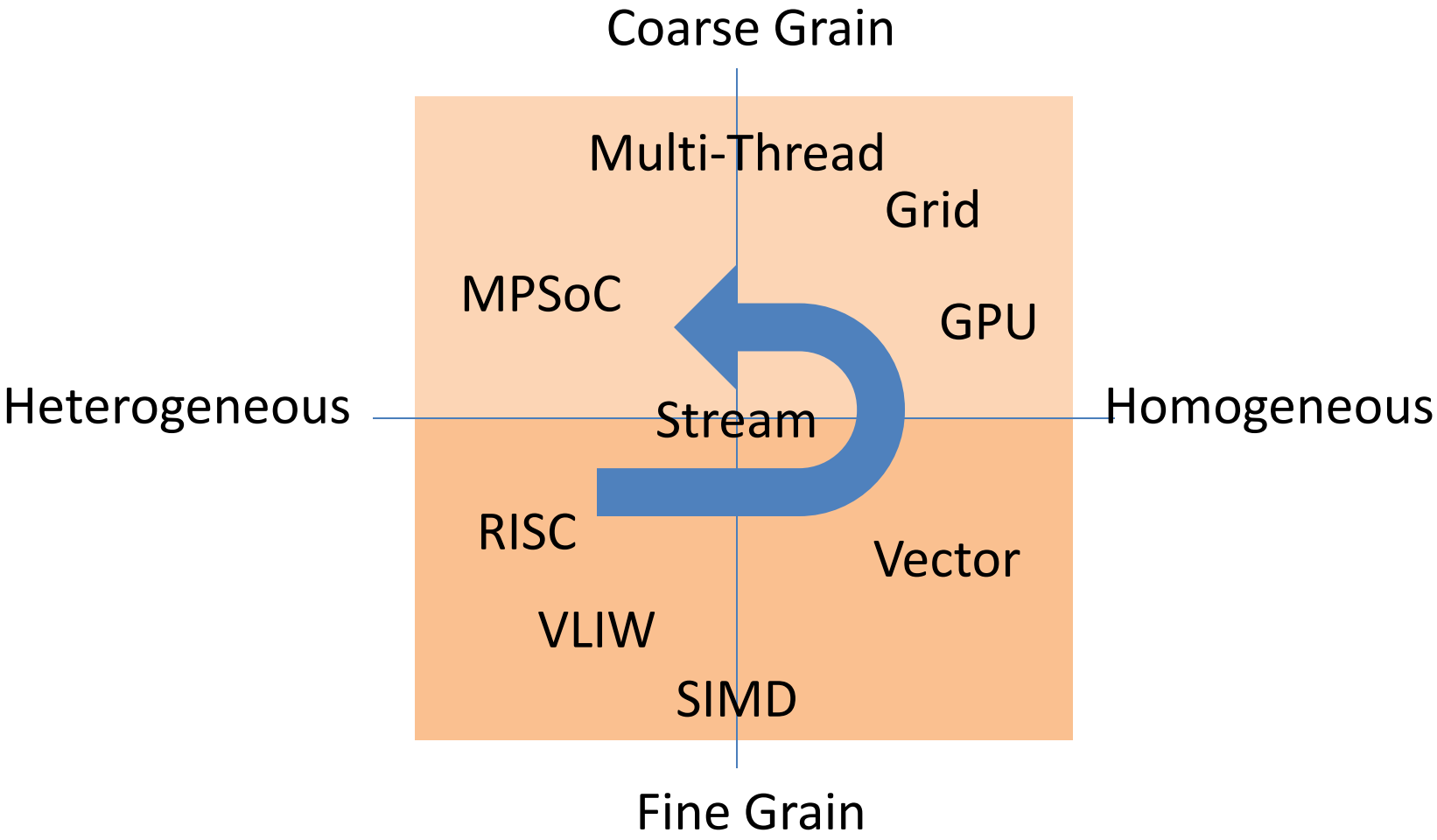
OpenCL

Assembly

TIOBE Programming Community Index



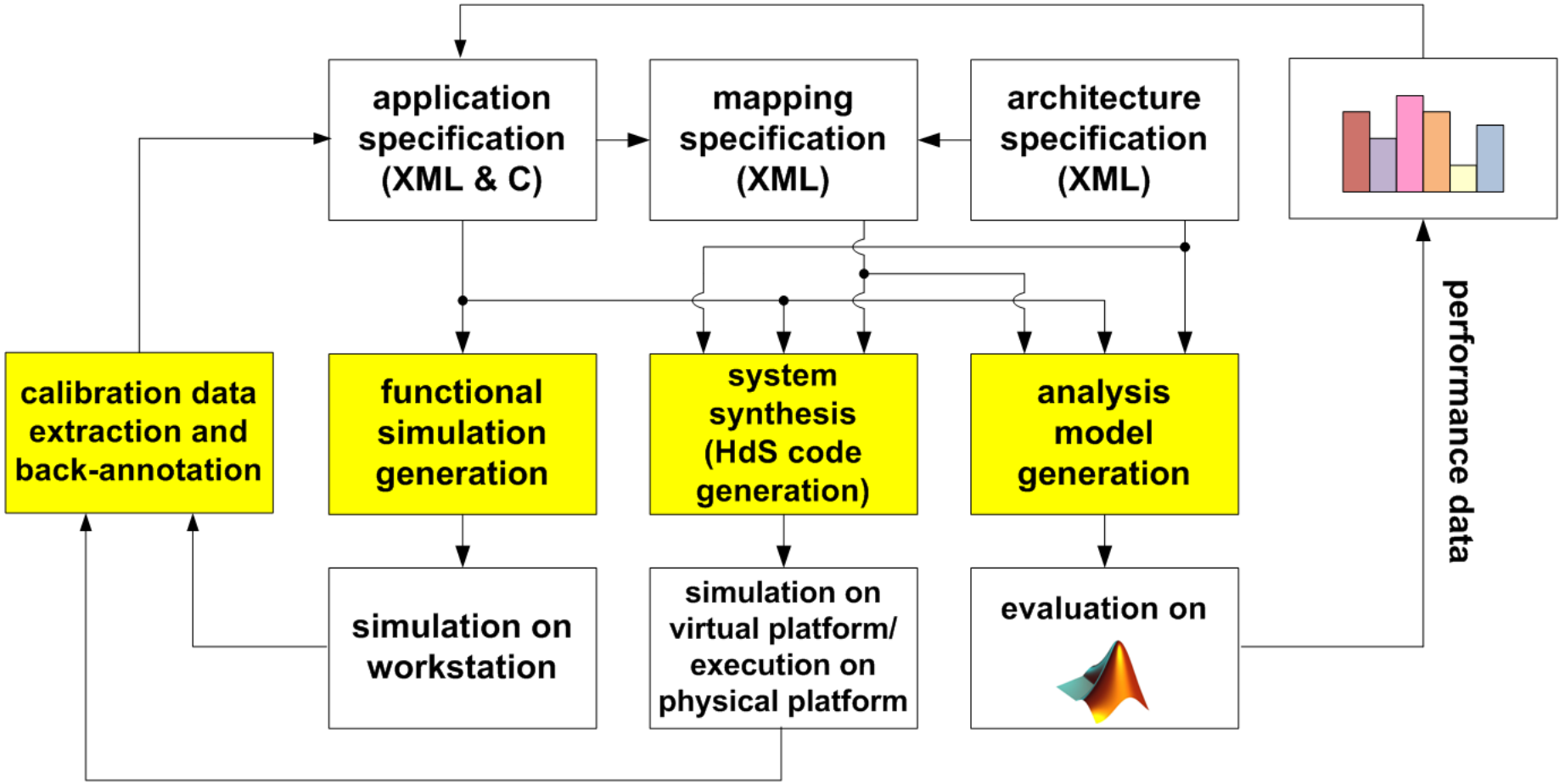
Parallel Architectures in Two Dimensions



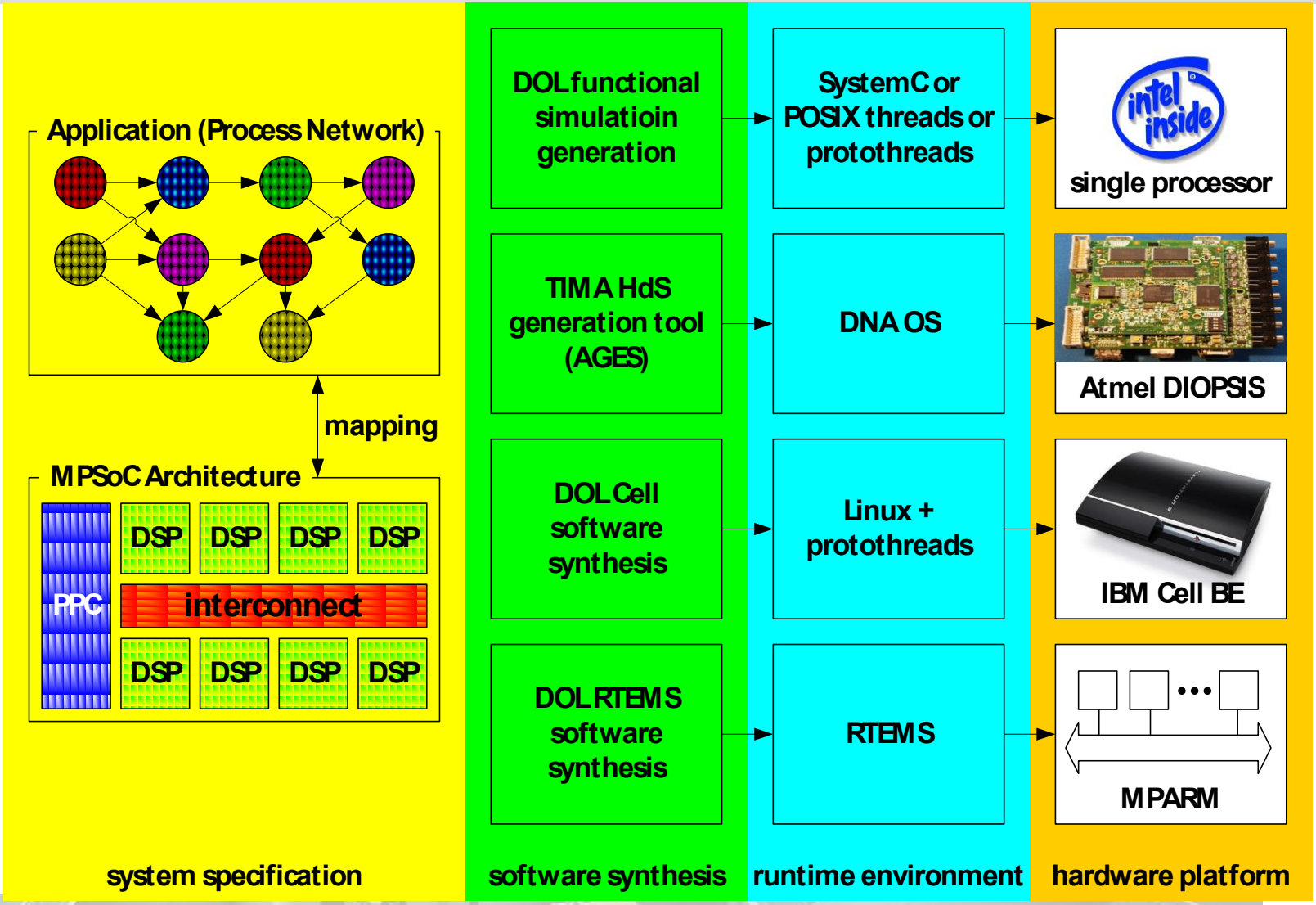
What We Need

- Coarse-grained level parallelism
- Design space exploration
- Easy to adopt Legacy code
- Non-functional prosperity verification

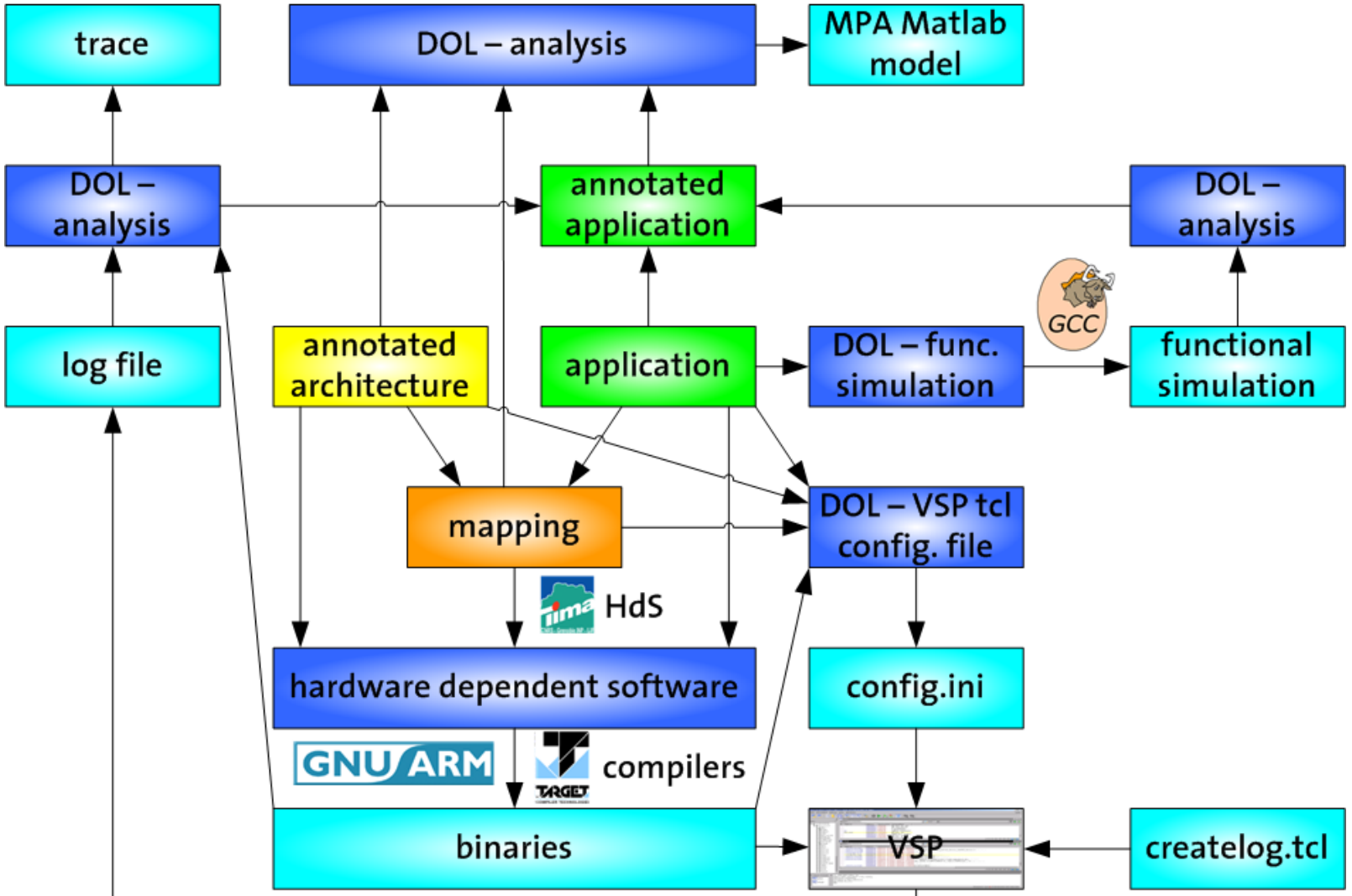
Distributed Operation Layer in ETHZ-TIK



Versatile MPSoC Software Design Flow



EU Project SHAPES Tool-Chain



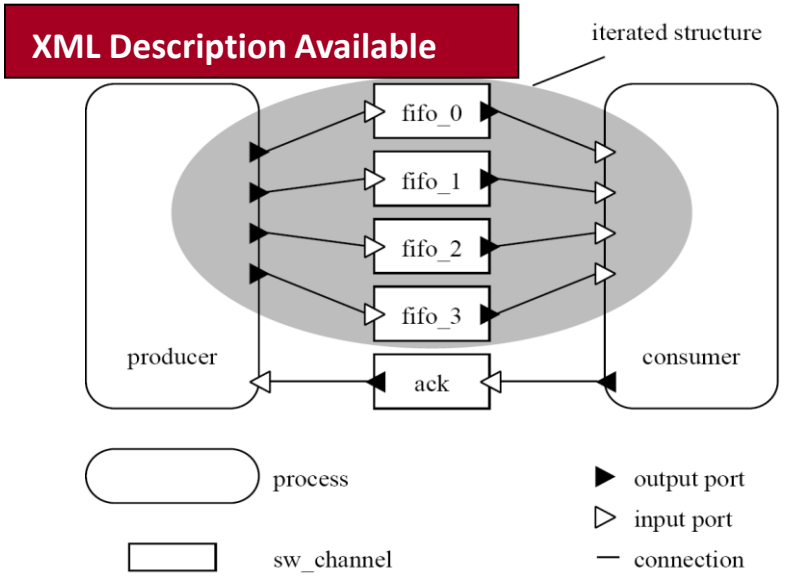
Application Specification

Structure

- **Process Network**
  - Processes
  - SW channels (FIFO behavior)
- **Iterators**
  - Scalability for processes, SW channels, entire structures

Functional specification

- **Language**: C/C++
- **API:** DOL primitives

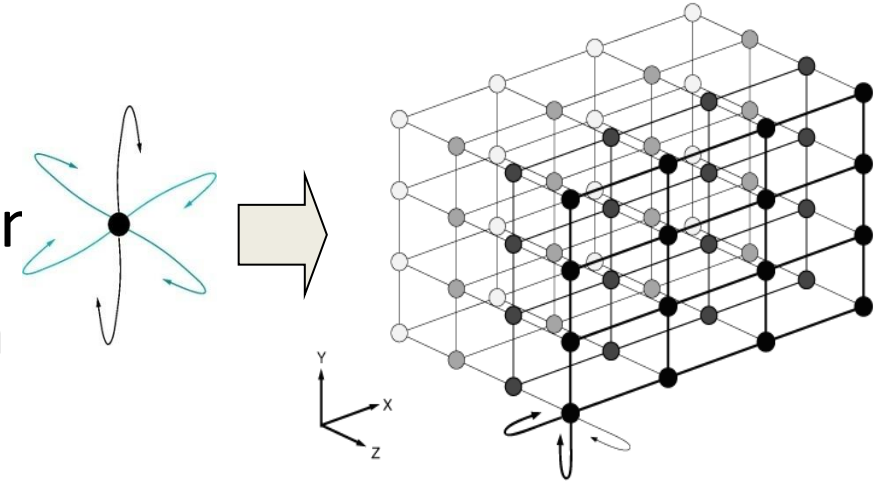


Algorithm 1 Process Model

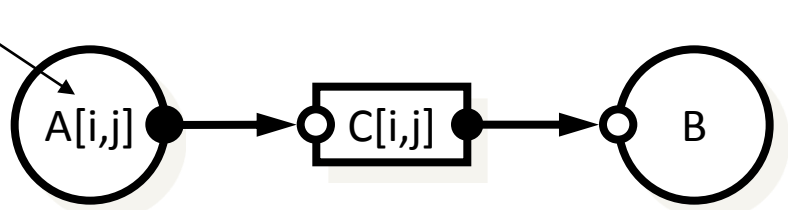
```
1: procedure INIT(DOLProcess p)                ▷ initialization
2:     initialize local data structures
3: end procedure

4: procedure FIRE(DOLProcess p)                ▷ execution
5:     DOL_read(INPUT, size, buf)              ▷ blocking read
6:     manipulate
7:     DOL_write(OUTPUT, size, buf)            ▷ blocking write
8: end procedure
```

Scalability at Specification Level

- Separation of instruction/thread level parallelism (inside processes) and process-level parallelism.
- Use of iterators in
  - architecture specificatior
  - application specification
  - mapping specification



$$\{(i,j) : 1 \leq i \leq N \wedge i \leq j \leq N\}$$

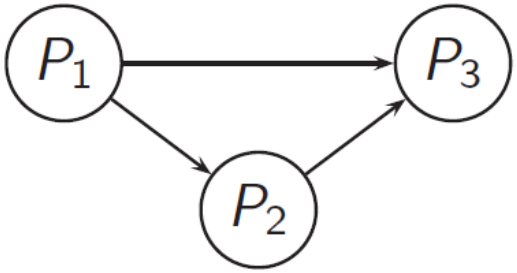


Definition of a process and Example

```
// Process.h: definition of a process

// local information
typedef struct _local_states *LocalState;
// function pointer
typedef void    (*ProcessInit)(struct _process*);
typedef int     (*ProcessFire)(struct _process*);
// place holder
typedef void    *WPTR;

typedef struct _process {
     LocalState      local;
     ProcessInit     init;
     ProcessFire     fire;
     WPTR            wptr;
} Process;
...
```

Definition of a process and Example

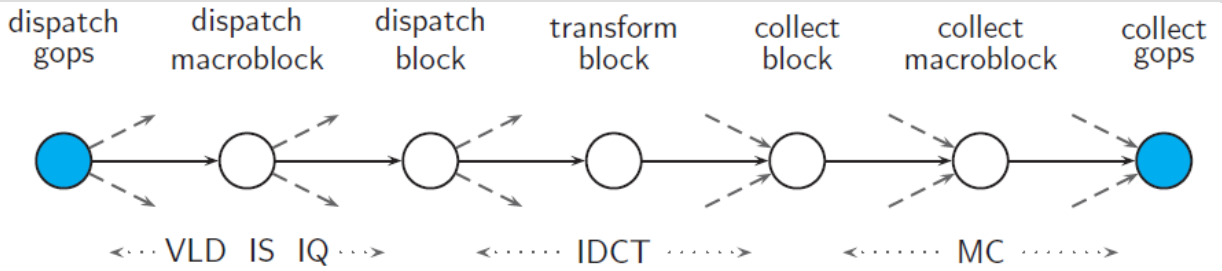
```
// P2.h: header file
typedef struct _local_states {
    int index;    int len;
} P2_State;

#define PORT_IN  1
#define PORT_OUT 2

// P2.c: C file
void P2_INIT(Process *p) {
    p->local->index = 0;
    p->local->len = LENGTH;
}

int P2_FIRE(Process *p) {
    float i;
    if (p->local->index < p->local->len) {
        READ((void*)PORT_IN, &i, sizeof(float), p);
        WRITE((void*)PORT_OUT, &i, sizeof(float), p);
        p->local->index++; }
    if (p->local->index >= p->local->len)
        DETACH(p);

    return 0;
}

// main.c: sample main function
int main() {
    Process p2; P2_State p2_state;
    p2.local = p2_state;  p2.init  = p2_INIT;  p2.fire  = p2_FIRE;
}
```



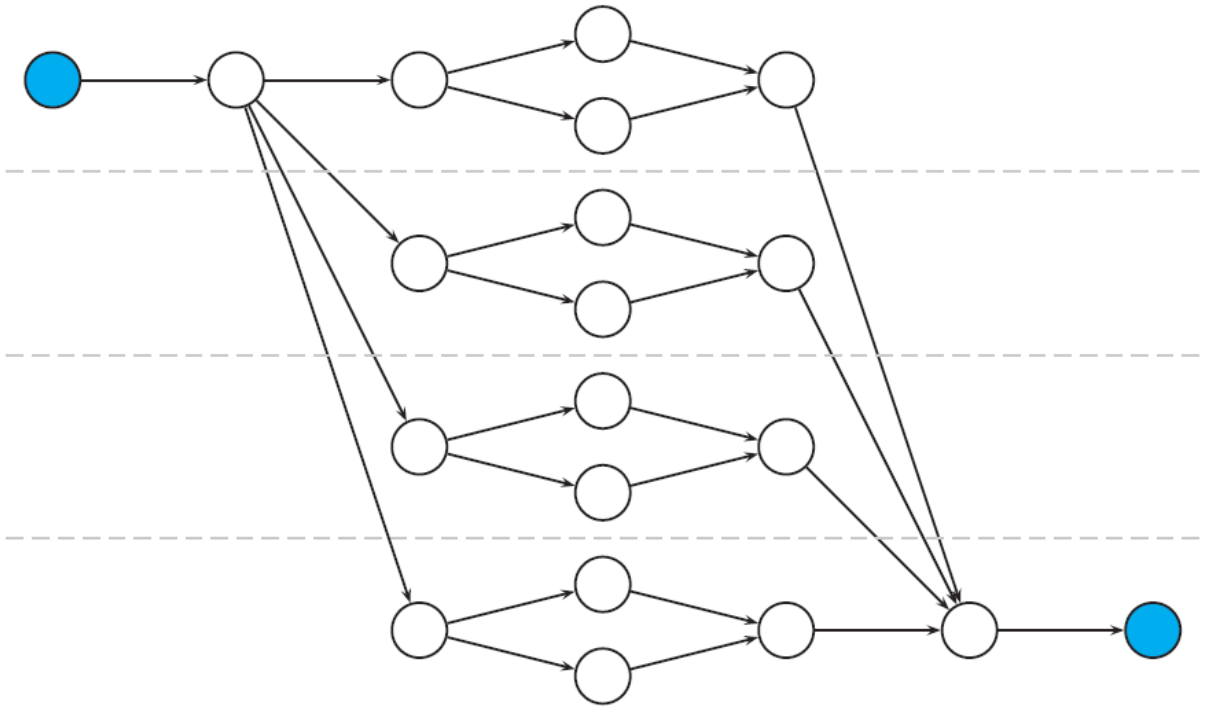
Scalable MPEG-2 Decoder

```
<!-- N1 is the number of gops processed in parallel -->
<variable name="N1" value="1" />
<!-- N2 is the number of macroblocks processed in parallel -
<variable name="N2" value="4" />
<!-- N3 is the number of blocks processed in parallel -->
<variable name="N3" value="2" />

<!-- instantiate processes -->
<process name="dispatch_gops">
  <iterator variable="i" range="N1">
    <port type="output" name="out" append function="i" />
  </iterator>
  <source type="c" location="dispatch_gops.c" />
</process>
...
<iterator variable="i" range="N1">
  <iterator variable="j" range="N2">
    <process name="dispatch_blocks">
      <append function="i" />
      <append function="j" />
      <iterator variable="k" range="N3">
        <port type="output" name="out"append function="k" />
...
```

Scalable MPEG-2 Decoder

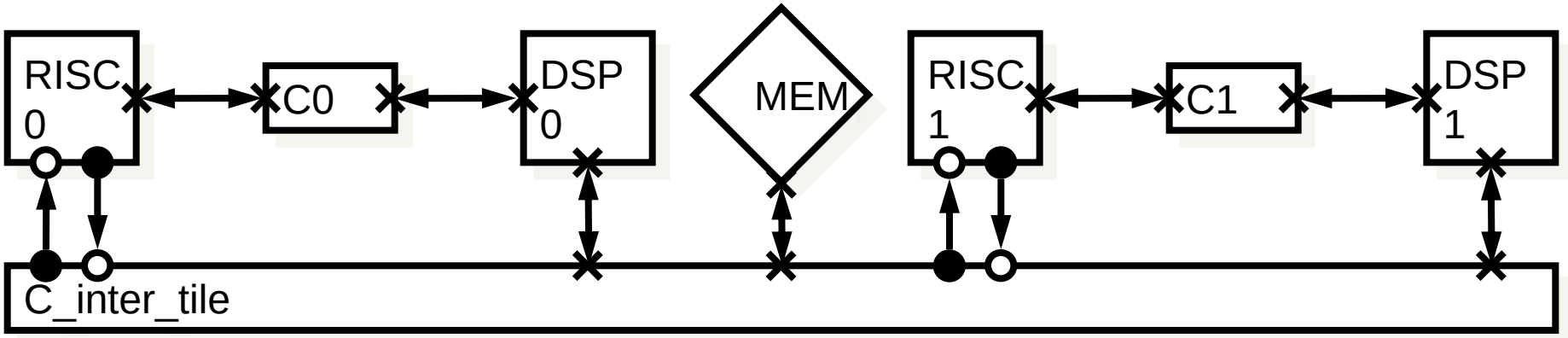


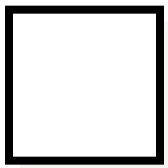
$N_1 = 1,\ N_2 = 1,\ N_3 = 1$



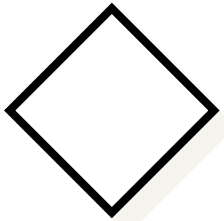
$N_1 = 1,\ N_2 = 4,\ N_3 = 2$

Target Platform Abstraction (1)





processor



memory

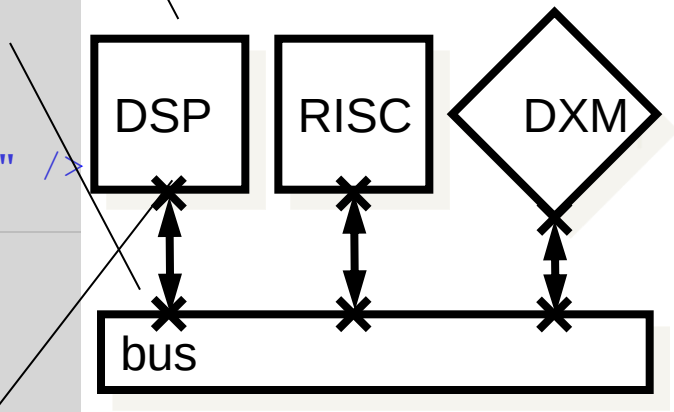
hardware channel

input / output / bidirectional port

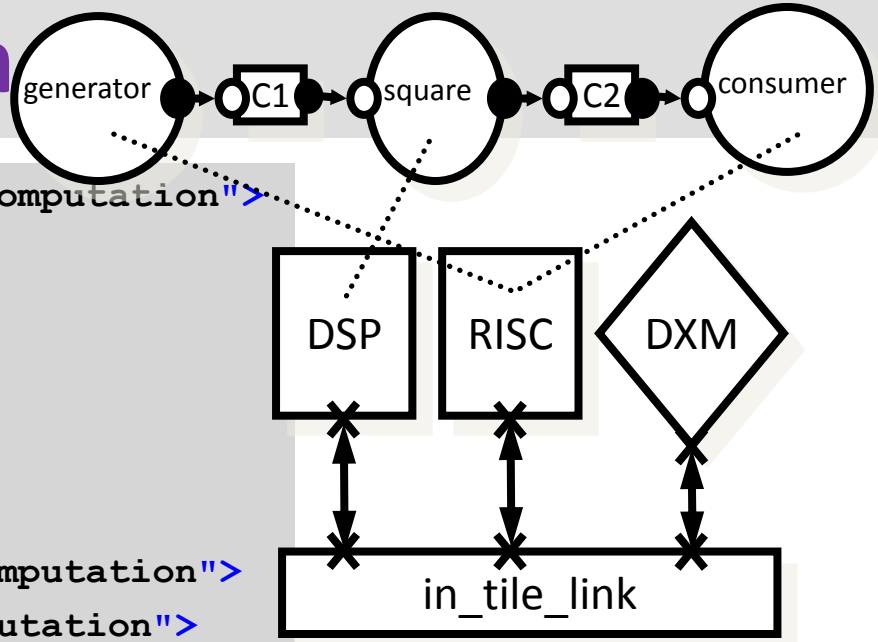
connection

Target Platform Abstraction (2)

```
- <processor name="processor1" type="DSP">
     <port name="processor_port" type="duplex" />
     <configuration name="clock" value="100 MHz" />
  </processor>
+ <processor name="processor2" type="RISC">
+ <memory name="sharedmemory" type="DXM">
- <hw_channel name="in_tile_link" type="bus">
     <port name="port1" type="duplex" />
     <port name="port2" type="duplex" />
     <port name="port3" type="duplex" />
     <configuration name="buswidth" value="32bit" />
  </hw_channel>
- <connection name="processor1link">
     <origin name="processor1">
        <port name="processor_port" />
     </origin>
     <target name="in_tile_link">
        <port name="port1" />
     </target>
  </connection>
+ <connection name="processor2link">
+ <connection name="memorylink">
```



Mapping Specification

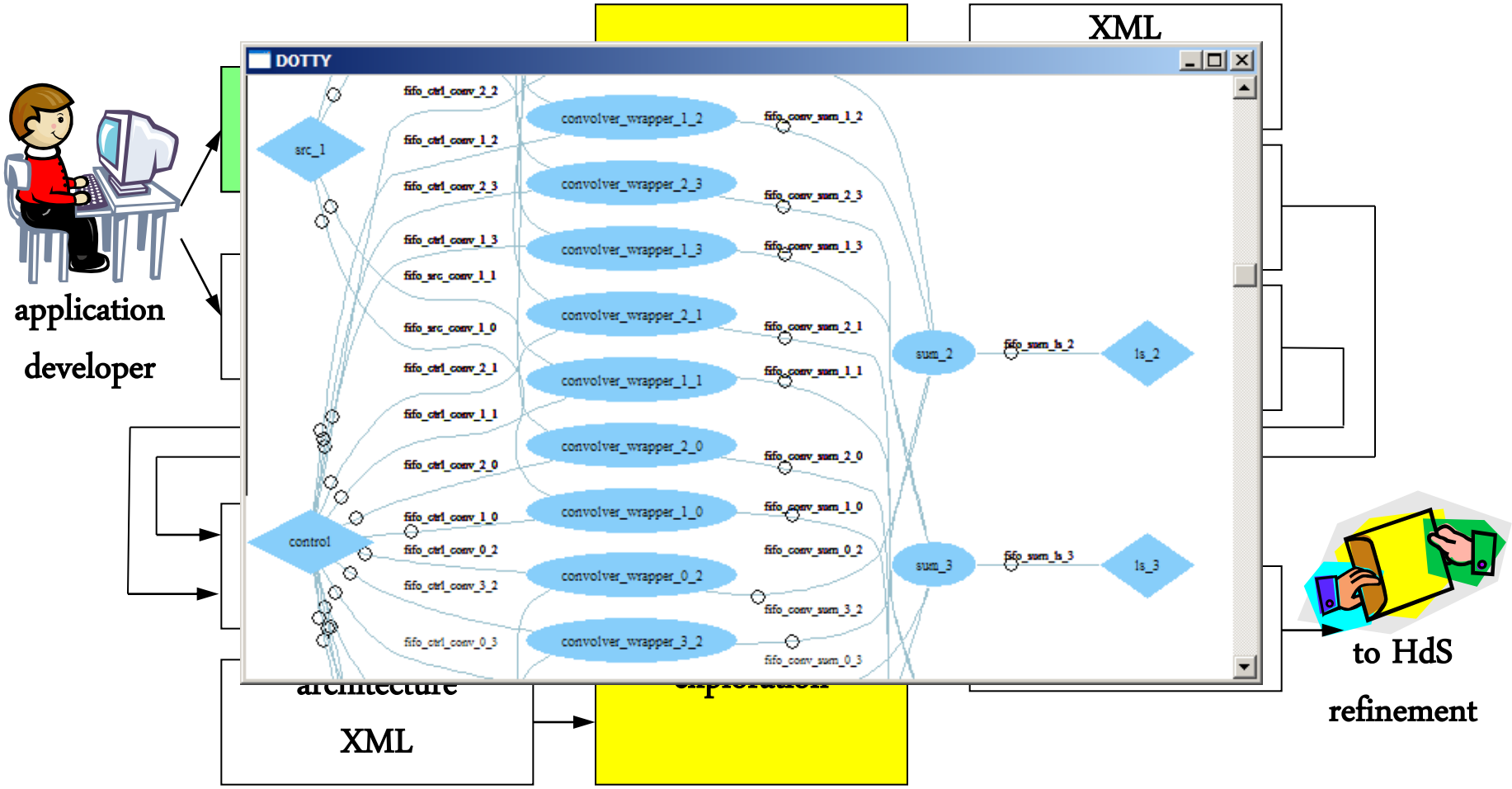



```
- <binding name="generator_binding" type="computation">
    <origin name="generator" />
    <target>
      <resource name="processor2" />
    </target>
    <schedule type="roundrobin" />
  </binding>
+ <binding name="consumer_binding" type="computation">
- <binding name="square_binding" type="computation">
    <origin name="square" />
    <target>
      <resource name="processor1" />
    </target>
    <schedule type="priorityfixed">
      <configuration name="priority" value="1" />
    </schedule>
  </binding>
+ <binding name="C1_binding" type="communication">
+ <binding name="C2_binding" type="communication">
```

Reasoning: Shift the design to system level and limit the ILP Wall to individual cores

Iterated mapping possible

DOL Design Flow Demonstration

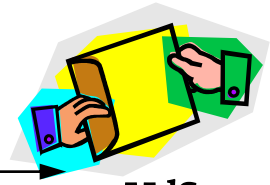


DOL Design Flow Demonstration

application developer

~/shapes/dolPrototype/trunk/build/bin/main

```
$ ./wfs/systemc/src/sc_application.exe

             SystemC 2.1.v1 --- Aug  3 2006 18:42:51
        Copyright (c) 1996-2005 by all Contributors
                    ALL RIGHTS RESERVED
src_0 = [];
src_0 = [src_0; 0.000000];
src_0 = [src_0; 0.000000];
src_0 = [src_0; 0.039260];
src_0 = [src_0; 0.078459];
src_0 = [src_0; 0.117537];
src_0 = [src_0; 0.156434];
src_0 = [src_0; 0.195090];
src_0 = [src_0; 0.233445];
src_0 = [src_0; 0.271440];
src_0 = [src_0; 0.309017];
src_0 = [src_0; 0.346117];
src_0 = [src_0; 0.382683];
src_0 = [src_0; 0.418660];
src_0 = [src_0; 0.453991];
src_0 = [src_0; 0.488621];
src_0 = [src_0; 0.522499];
src_0 = [src_0; 0.555570];
src_0 = [src_0; 0.587785];
src_0 = [src_0; 0.619094];
src_0 = [src_0; 0.649448];
src_0 = [src_0; 0.678801];
src_0 = [src_0; 0.707107];
src_0 = [src_0; 0.734323];
src_0 = [src_0; 0.760406];
src_0 = [src_0; 0.785317];
src_0 = [src_0; 0.809017];
src_0 = [src_0; 0.831470];
src_0 = [src_0; 0.852640];
src_0 = [src_0; 0.872496];
src_0 = [src_0; 0.891007];
src_0 = [src_0; 0.908143];
src_0 = [src_0; 0.923880];
src_1 = [];
src_1 = [src_1; 0.000000];
src_1 = [src_1; 0.000000];
src_1 = [src_1; 0.078459];
src_1 = [src_1; 0.156434];
src_1 = [src_1; 0.233445];
src_1 = [src_1; 0.309017];
```



to HdS refinement

DOL Design Flow Demonstration

application developer

process network XML

C source code

distributed operation layer

XML visualization

functional simulation

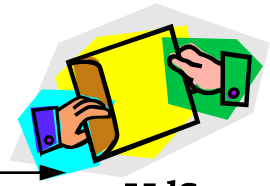
instr.-accurate simulation

ann. process network XML

architecture XML

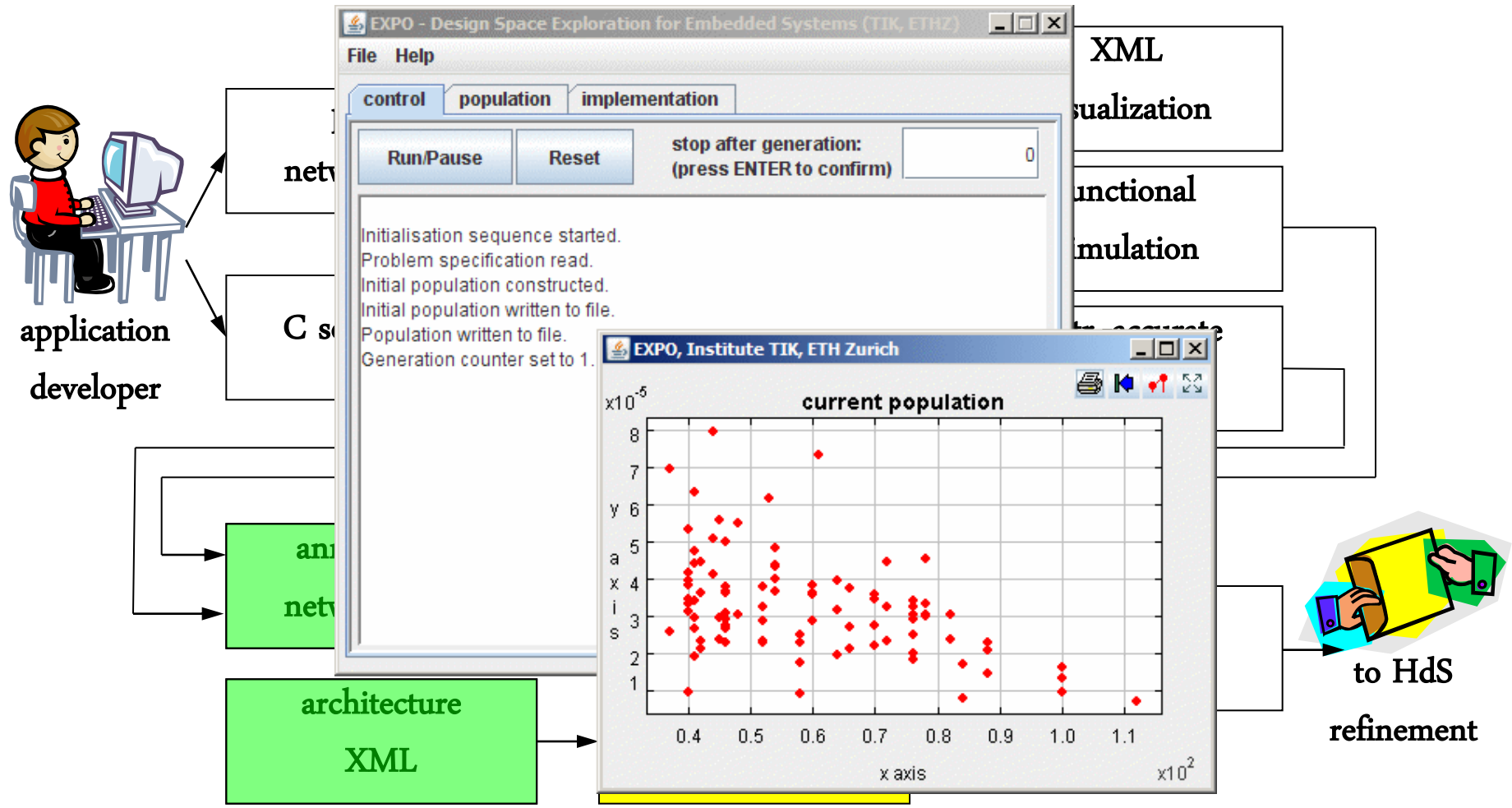
mapping design space exploration

mapping XML



to HdS refinement

DOL Design Flow Demonstration



Summary

- MPSoC programming
  - Reasons
  - Challenges
  - Principles
  - Practices