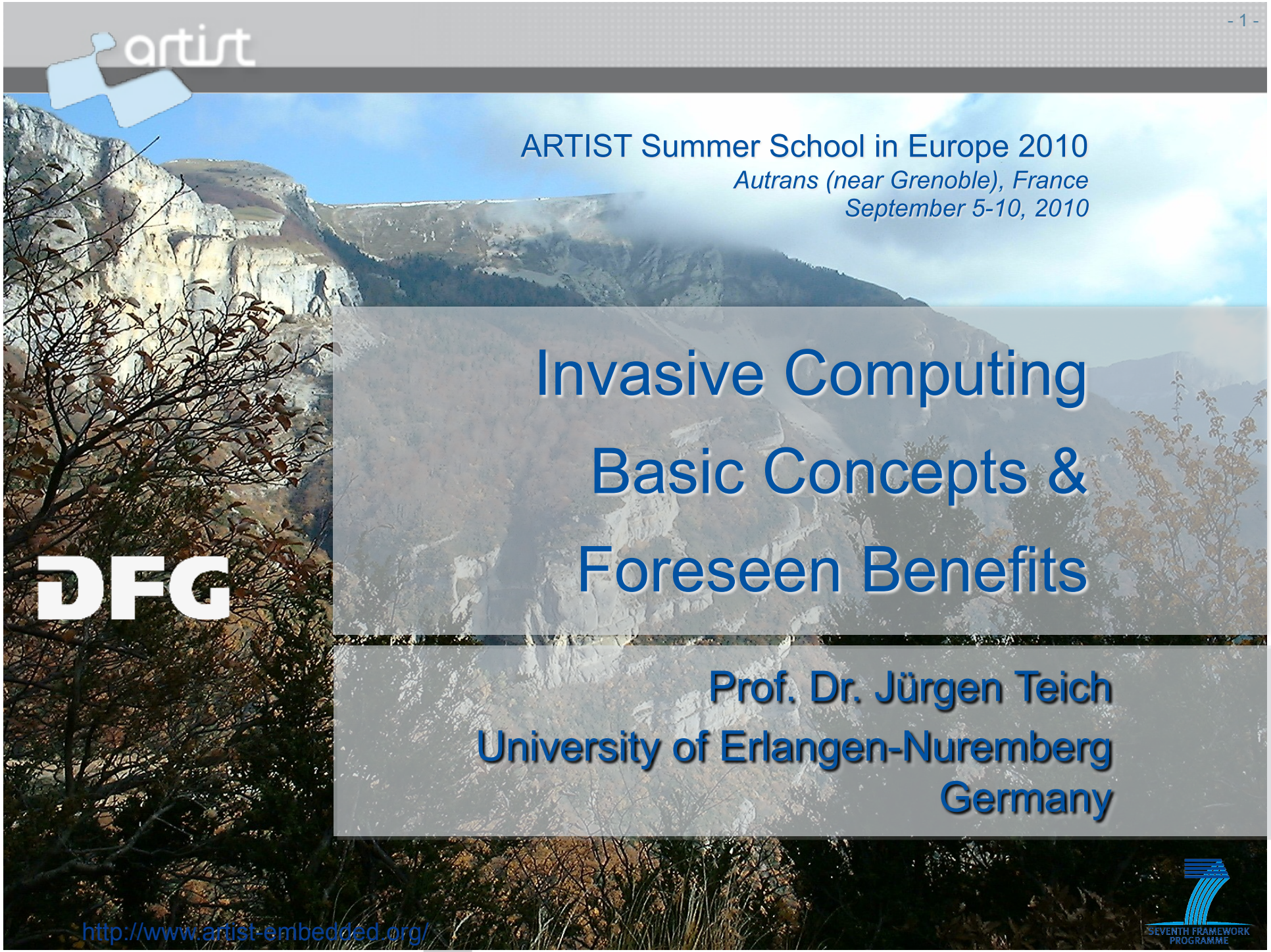


ARTIST Summer School in Europe 2010

Autrans (near Grenoble), France

September 5-10, 2010

Invasive Computing Basic Concepts & Foreseen Benefits

Prof. Dr. Jürgen Teich

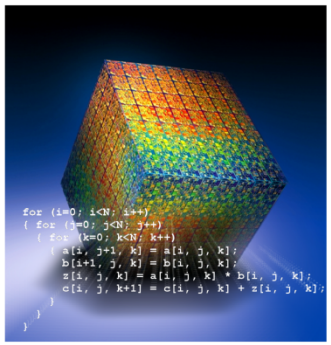
University of Erlangen-Nuremberg

Germany

Invasive Computing

Transregional Collaborative Research Centre 89

Friedrich-Alexander-Universität Erlangen-Nürnberg



Karlsruhe Institute of Technology

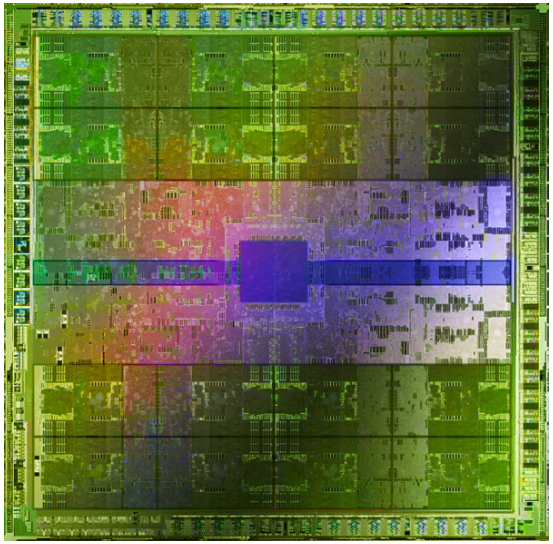
TRR 89

www.invasic.de

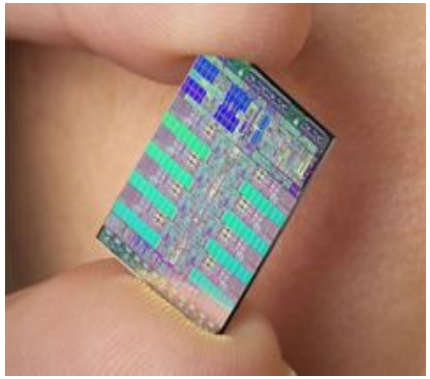
SEVENTH FRAMEWORK PROGRAMME

- What is Invasive Computing?
  - Uniquitousness of parallel computers
  - Challenges in the year 2020
  - Vision and Potentials
- Scientific Work Program
  - Basics: Resource-Aware Programming, Algorithms, Complexity
  - Architectures: Reconfigurability and Decentralized Resource Management
  - Tools: Compiler, Simulation Support and Run-Time System
  - Applications: Real-Time, Fault-tolerance, Efficiency and Utilization
- Structure, Chances and Goals
  - Project structure
  - Funded Institutions and Researchers
  - Demonstrator Roadmap
  - Impact and Risks

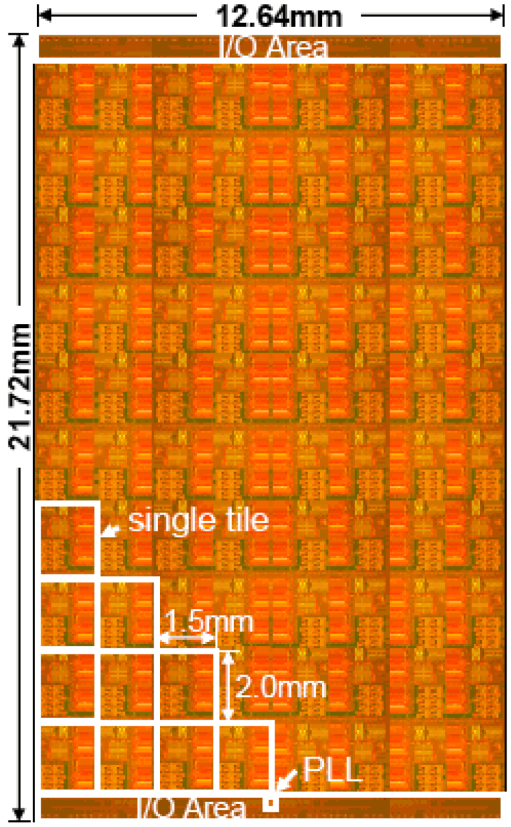
Ubiquitousness of parallel computers

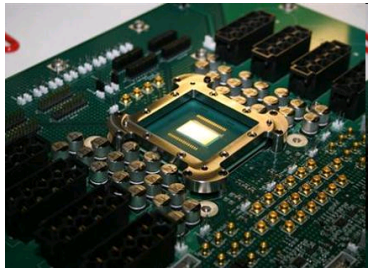


Nvidia Fermi: 512 Cores



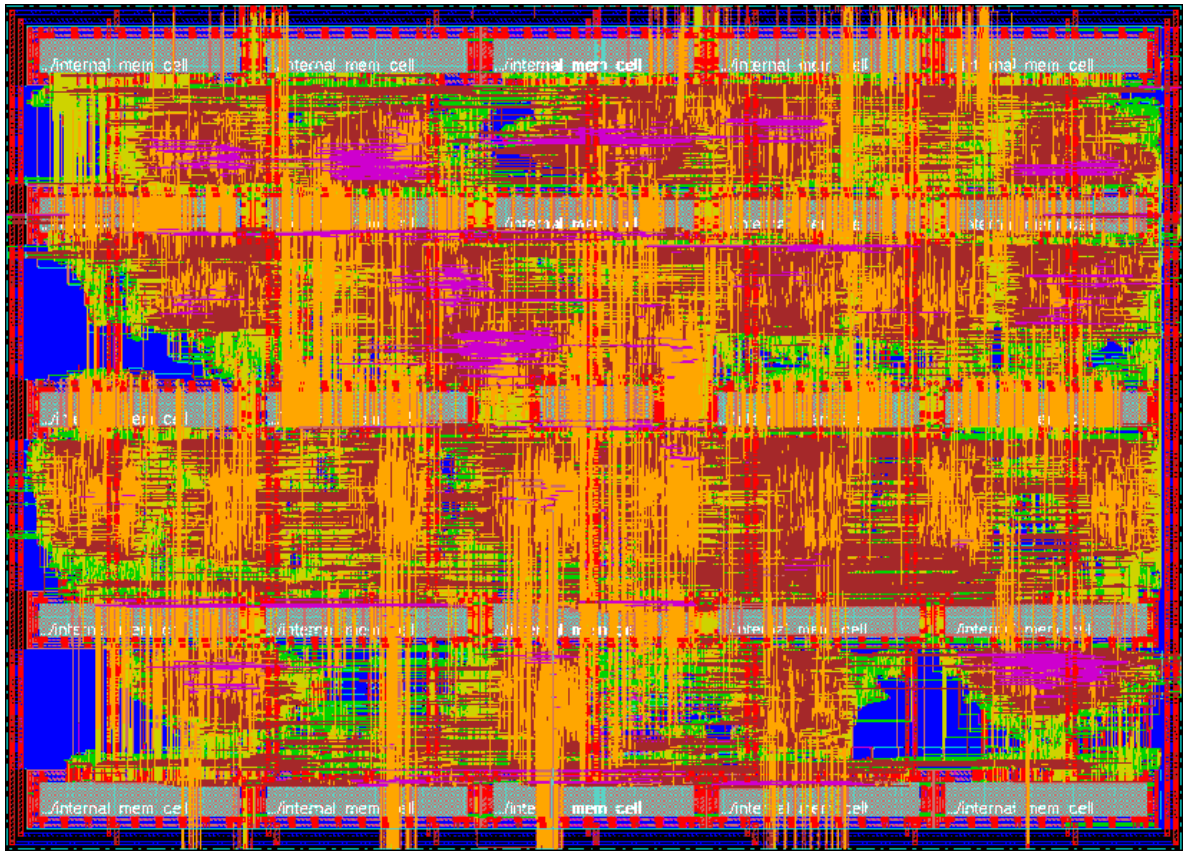
Sony Playstation 3,
IBM Cell 9 Cores





Intel Polaris: 80 cores

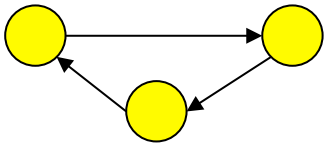
Ubiquitousness of parallel computers



Source: Hardware/Software Co-Design, Univ. of Erlangen-Nuremberg, Jan 2009. Programmable 5x5 core MPSoC for image filtering. Technology: CMOS 1.0 V, 9 metal Layers 90nm standard cell design. VLIW memory/PE: 16x128, FUs/PE: 2xAdd, 2xMul, 1xShift, 1xDPU. Registers/PE: 15. Register file/PE: 11 read/ 12 write ports. Configuration Memory: 1024x32 = 4KB. Operating frequency: 200 MHz. Peak Performance: 24 GOPS. Power consumption: 132,7 mW @ 200 MHz (hybrid clock gating). Power efficiency: 0,6 mW/MHz.

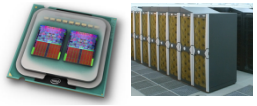
Abstraction Levels for Parallel Computations

process-level, thread-level



Hw + Sw Control

Multi Core



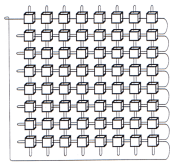
---

loop-level

```
FOR i=0 TO N DO
    FOR j=0 TO M DO
        ...
```

Hw-Ctrl.+ Func.

Processor Array



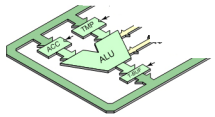
---

instruction-level

```
ADD R1, R2, R3
MUL R4, R1, $4
JMP $42
```

Hw-Ctrl. / VLIW

FUs



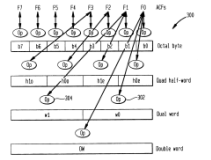
---

word-level, bit-level

```
01010001101010101010
10101010100011111111
```

Hw-Ctrl. / VLIW

SW-Units



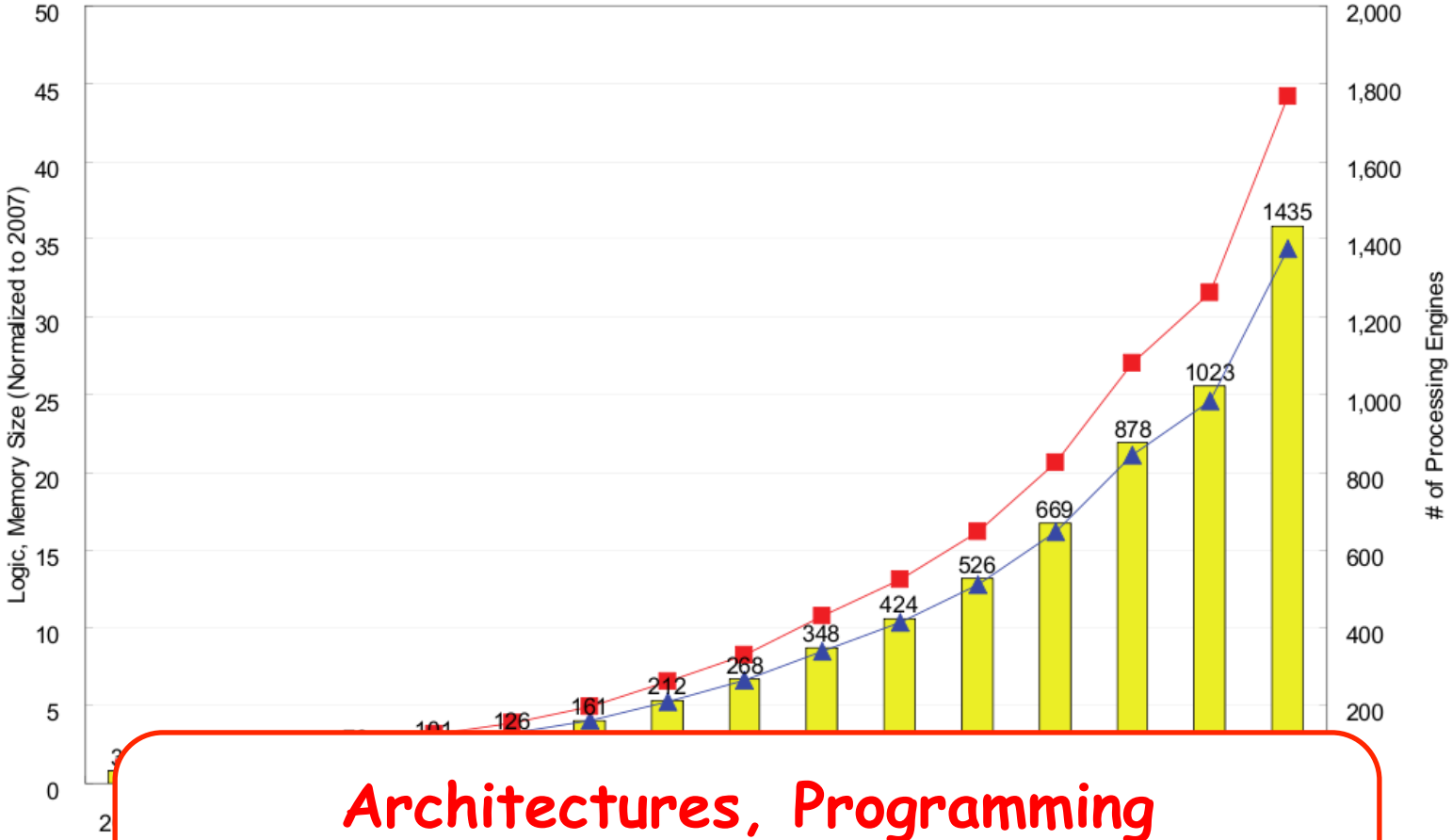
Exercise 1

- What kind of parallel computing systems and computing devices (i.e., MPSoCs are you aware of/have you already experience with?

  (3 minutes)

Name	Target class of apps./supported parallelism levels	Individual Experience (y/n)
Nvidia Tesla GPU	Computer Graphics Apps. Data/Thread	yes
Tilera Tile-GX 100 core	Signal Processing Fine Grain Parallel	no
Intel SCC	…	…
…	…	…

Challenges in the year 2020



Architectures, Programming and Management of Applications for 1000s of Processors in 2020?

Challenges in the year 2020

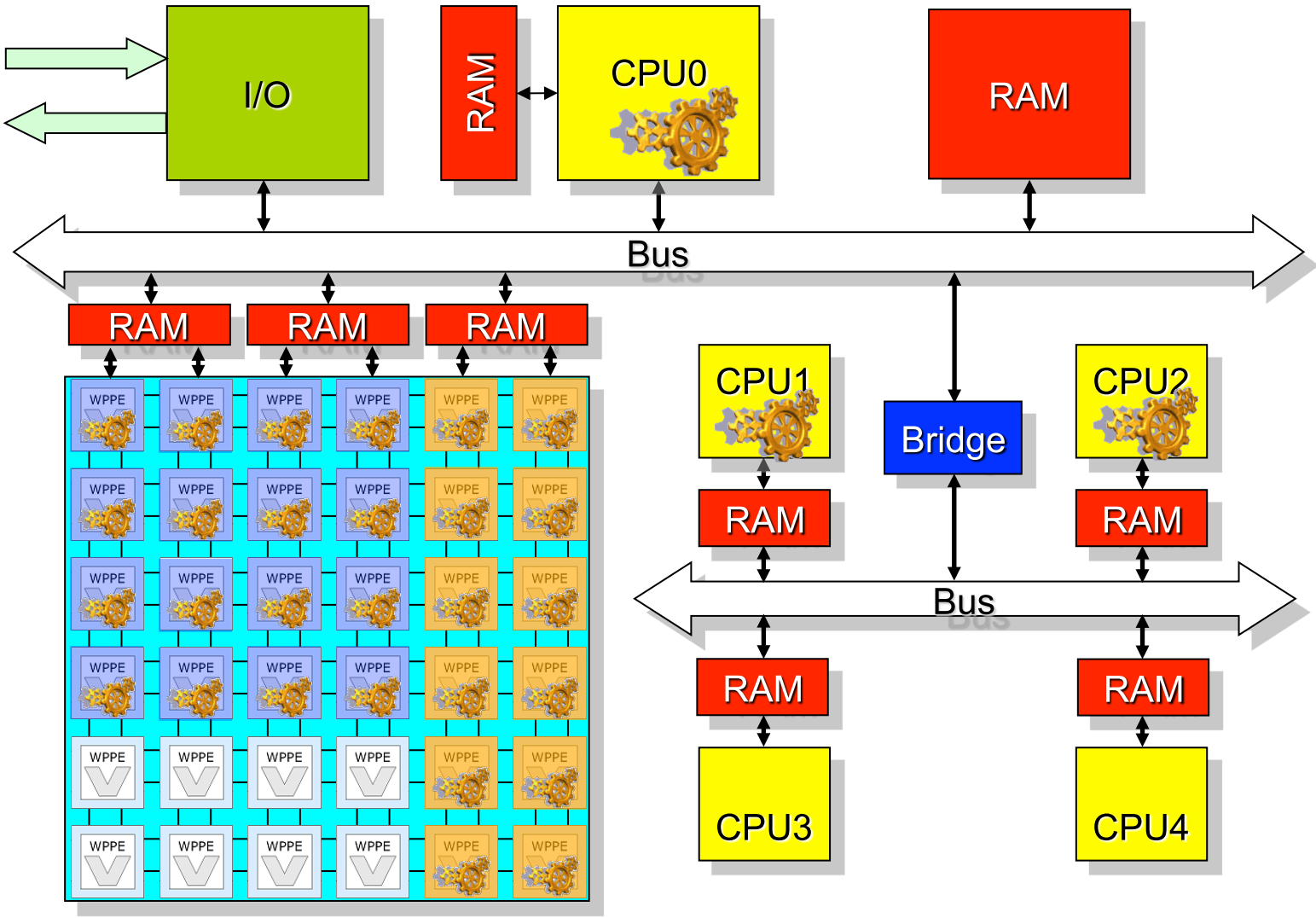
- Complexity
  - How to map dynamically applications onto 1000 or more processors while considering memory, communication and computing resource constraints?
- Adaptivity
  - How and to what degree shall algorithms and architectures be adaptable (HW/SW, bit/word/loop/thread/process-level)?
- Scalability
  - How to specify and/or generate programs that may rund without (great) modifications on either 1,2,4, or N processors?
- Physical Constratins
  - Low power, performance exploitation, management overhead
- Reliability and Fault-Tolerance
  - Necessity for compensation of process variations as well as temporal and permanent defects

Invasive Computing

- Definition
  [J. Teich. Invasive Algorithms and Architectures.
  Journal it – Information Technology, 50 (2008),
  pp. 300-310, 2008]

Invasive Programming denotes the capability of a program running on a parallel computer to request and temporarily claim processing, communication and memory resources in the neighborhood of its actual computing environment, to then execute in parallel using these claimed resources, and to be capable to subsequently release these resources again.

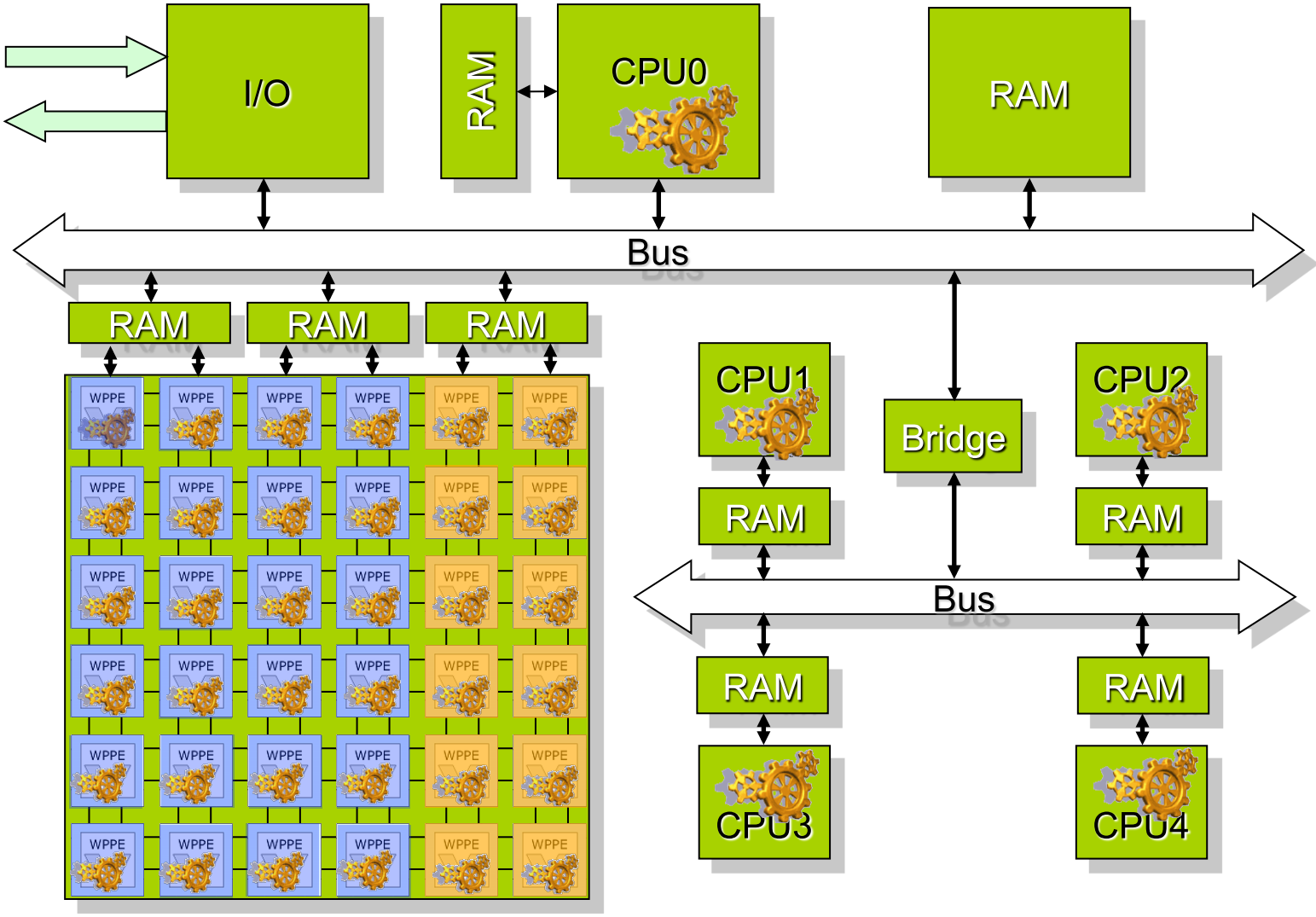
Invasion: Example



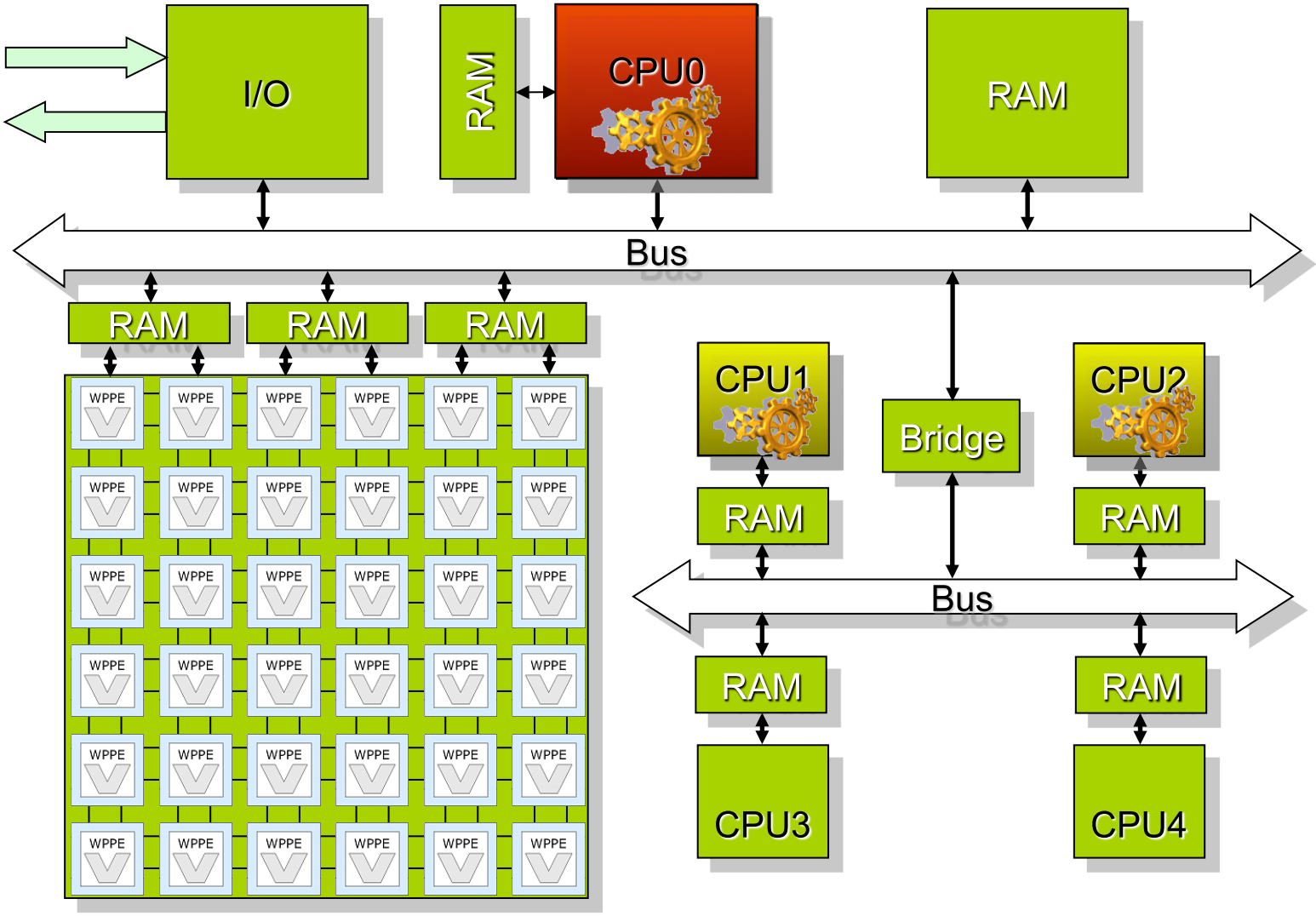
Vision and Potentials

- Run-Time Scalability
  - Today´s parallel programs are in general not able to adapt themselves to the current availablity of resources.
  - Today´s computer architectures do not support any application-controlled resource reservation.
- Dynamic Self-Optimization possible through Invasion wrt.
  - Ressource Utilization
  - Power Consumption (Temperature Management)
  - Performance
- Tolerance of Failures and Defects
  - Today´s parallel programs just would not run (correctly) any more!
- Robustness
  - Applications tolerate a variable availability of resources

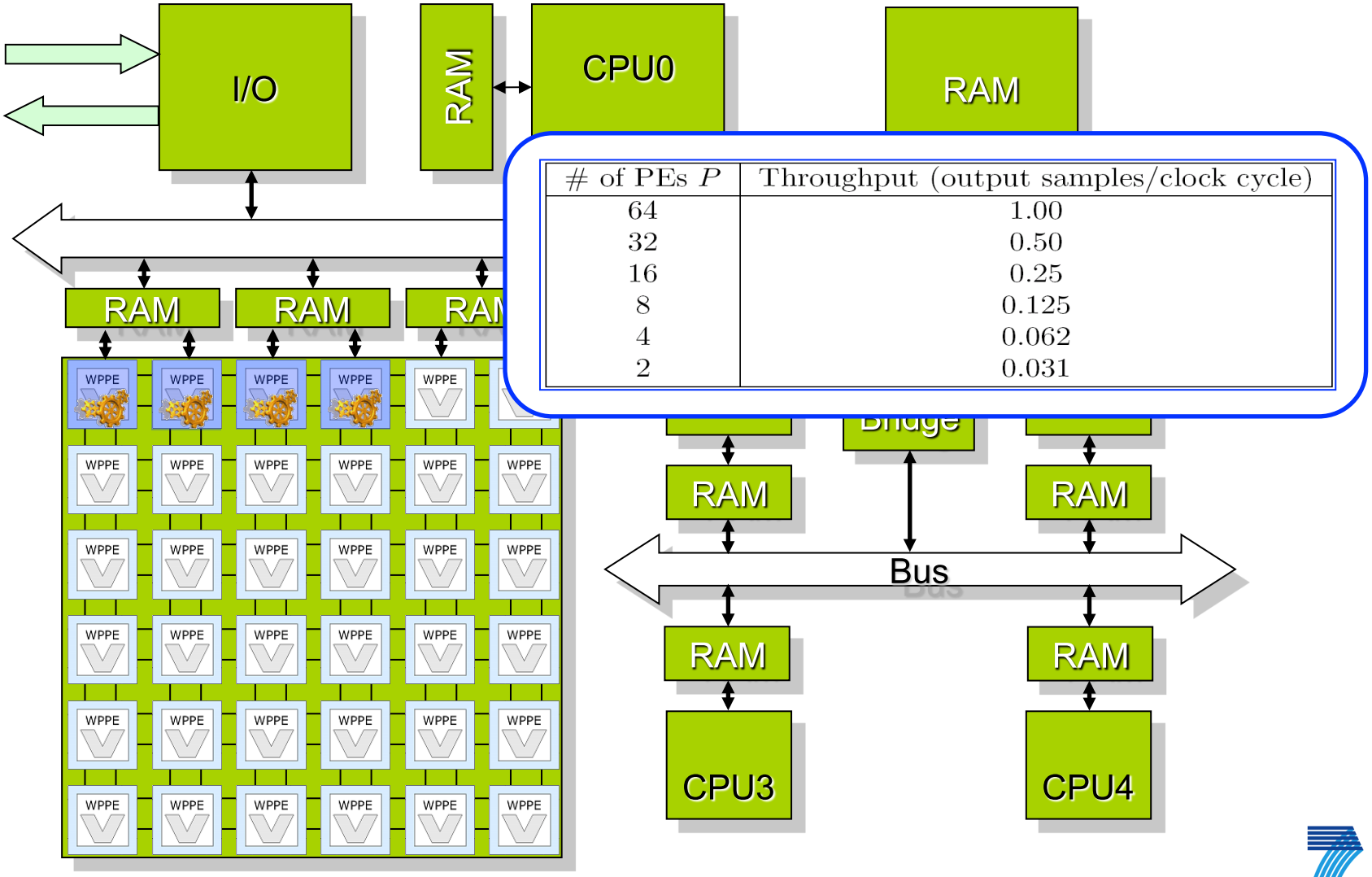
Potential: Resource Utilizations up to 100%



Potential: Power and Temp. Management

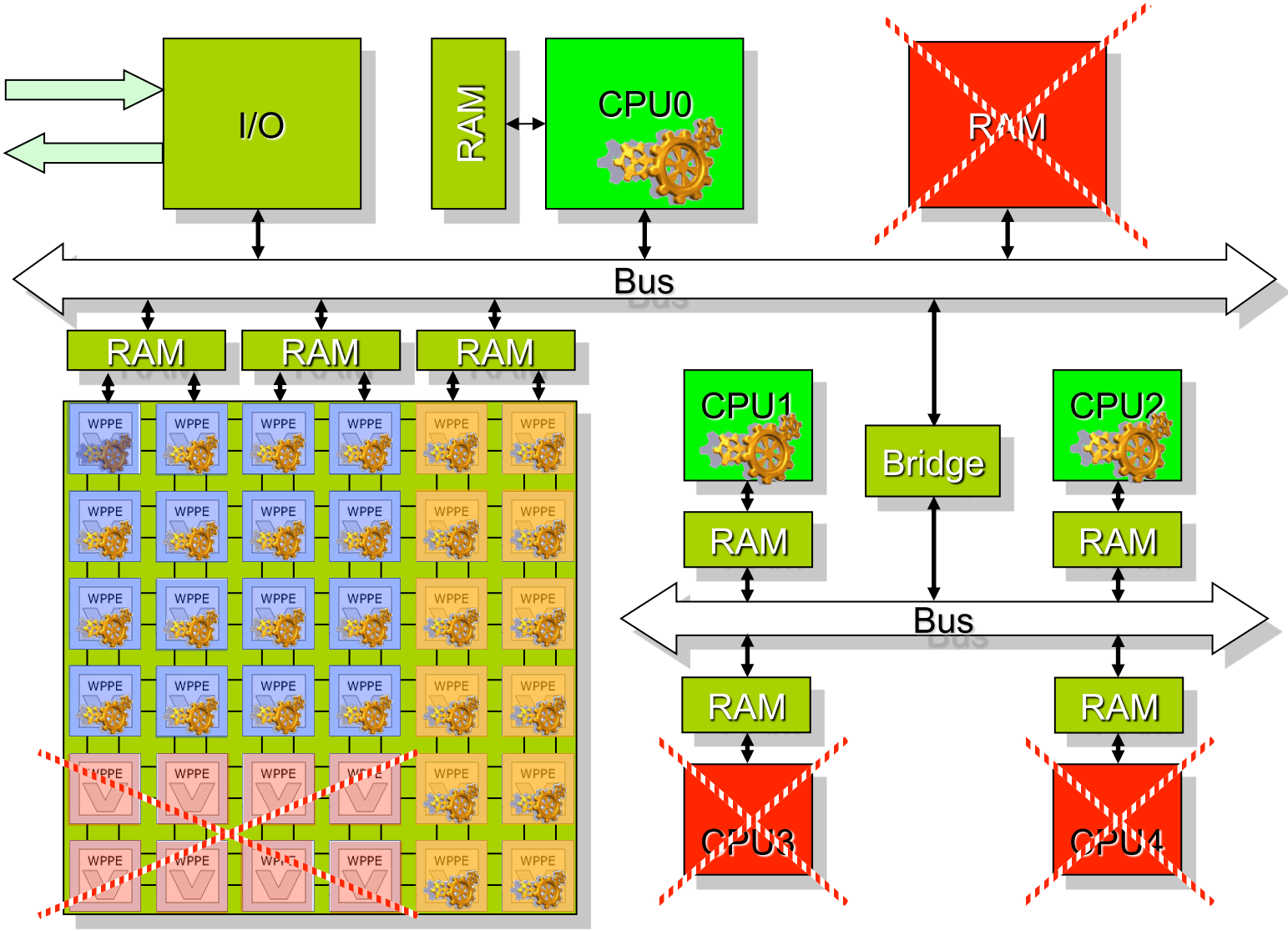


Potential: Performance Gain/Tradeoff



# of PEs P	Throughput (output samples/clock cycle)
64	1.00
32	0.50
16	0.25
8	0.125
4	0.062
2	0.031

Potential: Robustness and Fault-Tolerance



Exercise 2

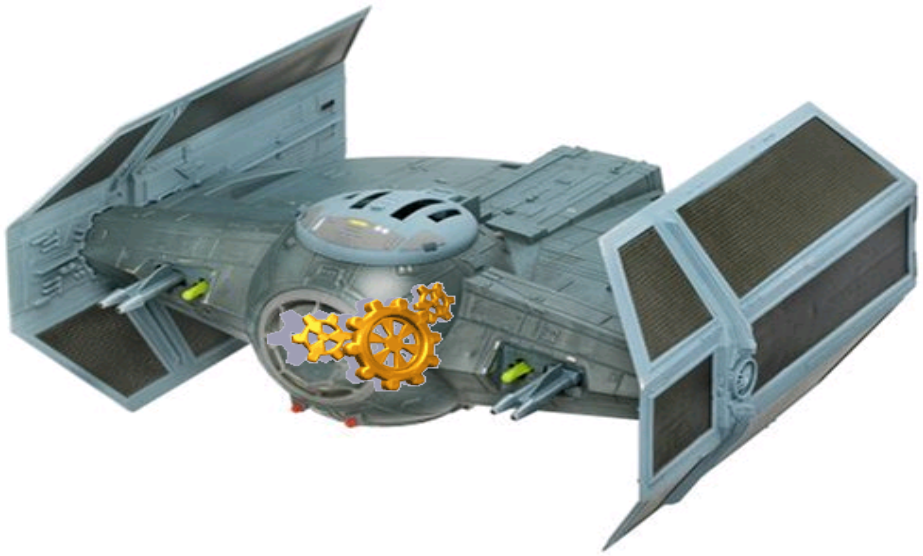
- A) Explain the term, respectively main idea of “INVASIVE COMPUTING” in your own words
- B) List the major advantages and potentials of “INVASIVE COMPUTING”
- C) Imagine and name similarities and differences to recent research areas such as “GRID COMPUTING” and “CLOUD COMPUTING”.

(5 minutes)

- What is Invasive Computing?
  - Uniquitousness of parallel computers
  - Challenges in the year 2020
  - Vision and Potentials
- Scientific Work Program
  - Basics: Resource-Aware Programming, Algorithms, Complexity
  - Architectures: Reconfigurability and Decentralized Resource Management
  - Tools: Compiler, Simulation Support and Run-Time System
  - Applications: Real-Time, Fault-tolerance, Efficiency and Utilization
- Structure, Chances and Goals
  - Project structure
  - Funded Institutions and Researchers
  - Demonstrator Roadmap
  - Impact and Risks

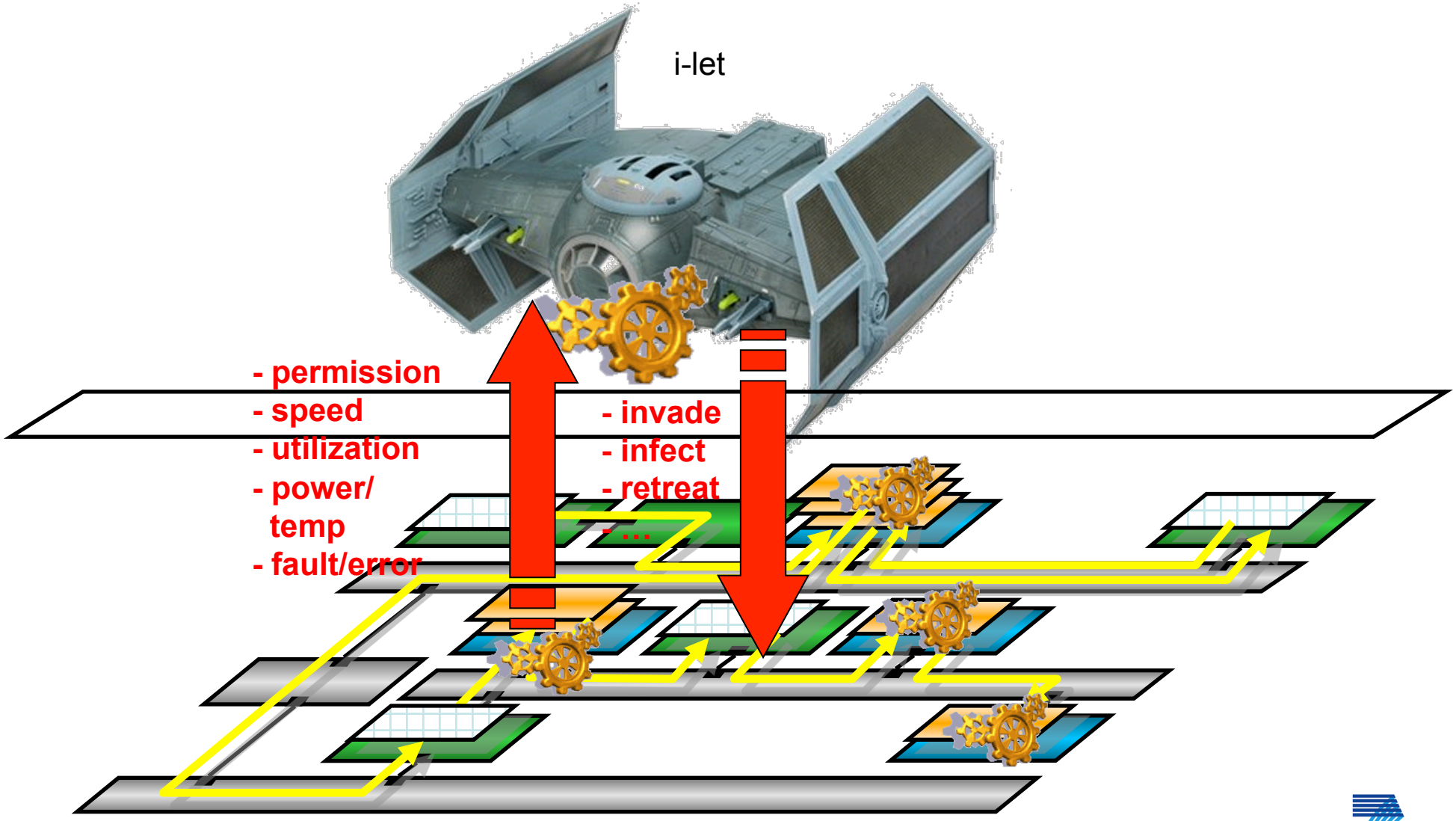
Basic Functionality of Invasive Programs

- Invade
  Construct(s) for request and reservation of resources (processors, memory, interconnect)
- Infect
  Construct(s) for programming, resp. configuration of resources (processors, memory, interconnect) for special services
- Retreat
  Construct(s) for release of resources (processors, memory, interconnect)



Concept invade-let (i-let)

Basics of Invasive Programming



Exercise 3

- A) Why should an application be aware of the operating state of the underlying hardware resources?
  Give at least 3 reasons.
- B) Why can´t or shouldn´t resource-aware programming be provided just by the middleware/OS and thus be hidden to the application programmer? What are advantages of resource consideration at the algorithmic or application-programmer´s level?
- C) What advantages/disadvantages or risks may resource-aware programming have or introduce?

Advantages	Disadvantages
Fault-Tolerance	Overheads (HW, SW)
Resource Efficiency	Speed?
…	…

(5 minutes)

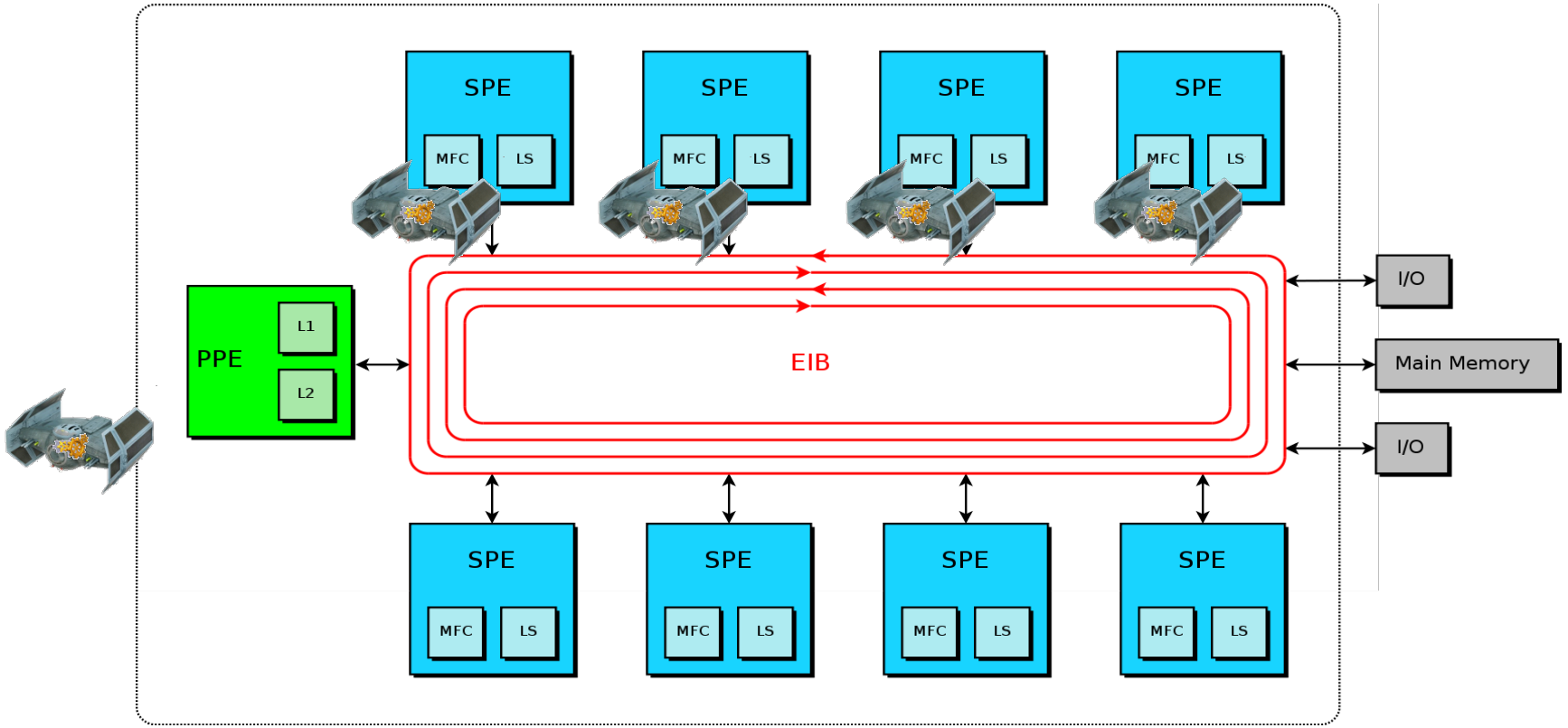
Invasive Computing on MPSoCs

- Question:
  Is invasive computing already possible on MPSoC platforms available today?

- First case study: Cell B.E.
  - Library-based implementation of basic functionality
  - PPE: “Master of Invasion”
  - SPE: Sorting as slave
  - Algorithm: Invasive implementation of well-known bitonic sorter for exploitation of available SIMD architecture
  - Merging accomplished by PPE

Cell B.E. Architecture

- 8 Synergistic Processing Elements (SPEs)
- 1 Power Processing Element (PPE)
- Element Interconnect Bus (EIB): ring bus → no direct neighbour



Library-Based Approach

- Link your invasive program with library:
  - `ppu-g++ -o … -linvasic …`
- Library `invasic` provides all basic commands:
  - SPE initialization: `init_spus(program)`
  - Invade command: `spe_struct_t *invade(uint32_t num_spe)`
  - Infect command: `void infect(spe_struct_t *spes, workload_struct_t *workloads, uint32_t signal)`
  - Retreat command: `void retreat(spe_struct_t *spe)`
  - SPE cleanup: `exit_spus()`
- Planned: Support of distinct programs in each SPE
  - E.g., using code overlay techniques
  - Program will be loaded “on demand”
  - Main program on PPE runs always (Routine main)

Example code: Invasive Sorting

```
n = ... /* determine number of SPEs needed for sorting */
  /* invade */
  sort_spes = invade(n);
  if (sort_spes->num_spe == 0) {
      fprintf(stdout, "Couldn't invade any SPE, using serial version ...");
      sort_serial(...);
  } else {
      /* create workloads for each SPE; command not provided by library;
         distributes the work equally between all invaded SPEs;
         sets ea_in, ea_out and size of data */
     workloads = create_workloads(sort_spes, in_data, out_data, size);
      /* infect */
      infect(sort_spes, workloads, SPU_SORT);
      /* wait until all SPUs have finished (retreat) */
      for (i=0; i<sort_spes->num_spe; i++) {
          sort_spes->spes[i]->wait_mbox();
      }
  }
  /* merge data on PPE */
```

Project Area A – Basics

- Programming and Language Issues:
  - Finding and classification of elementary (basic) constructs for invasive programs (the *invasive command space*) [A1]
  - Definition of an abstract kernel language (syntax, semantics, type system) [A1]
  - Embedding of command set into programming language(s) [A1]
- Mathematical Models for Effifiency and Utilization Analysis of invasive applications [A1]
- Algorithm Engineering:
  - Complexity and cost invasive algorithms [A1]
  - Scheduling and Load Balancing [A3]

Exercise 4

- A) What kind of parallel programming languages, language extensions and/or programming environments such as APIs are you aware of?

Name	Language/ API	Architecture/ Application Target Class	Advantages (+)/ Disadvantages (-)
OpenMP	API for Fortran and C	Shared-Mem. Architectures/ Thread-level	(+) Wide spread (+) Quasi-standard (-) Shared-Mem. Archs. only
…	…		
…	…		

(3 minutes)

Parallel Programming Survey

	Tiling		Memory Architecture		Availability		Consortium	Approach				Load Balancing		Architectures		
	auto	user	shared	distributed	Commercial	OSS		Language	Library	Compiler	Comp. Ext.	SW	HW	x86	GPUs	others
OpenMP	✓	✓	✓				✓		(✓)		✓	✓		✓		✓
MPI		✓	(✓)	✓			✓		✓			✓		✓		✓
PGI	✓	✓	✓	✓	✓					✓		✓	(✓)	✓	(✓)	
TBB	✓	✓	✓		✓	✓			✓			✓		✓		✓
Cilk++	✓	✓	✓		✓	(✓)		✓		✓	(✓)	✓		✓		
CUDA		✓	✓	✓	✓			✓		✓			✓	(✓)	✓	
Brook+		✓	✓	✓	✓	(✓)		✓		✓			✓		✓	
OpenCL	✓	✓	✓	✓			✓	✓		✓		✓	✓	✓	✓	✓
RapidMind	✓	✓	✓	✓	✓				✓			✓	✓	✓	✓	✓
Ct	✓	✓	✓	✓	✓				✓			✓	✓	✓	✓	✓
Axum		✓		✓	✓			✓		✓		✓		✓		
X10	(✓)	✓	✓	✓		✓		✓		✓		✓		✓	(✓)	✓

The X10 Programming Language

- What is X10?
  - X10 is a new programming language being developed at IBM Research in collaboration with academic partners.
  - The development started in 2004 within the IBM PERCS project as part of the DARPA program on High Productivity Computing Systems (HPCS).
  - Designed for parallel programming using the partitioned global address space (PGAS) model.

- Features of X10
  - Is more productive than current models
  - Can support high levels of abstraction
  - Can exploit multiple levels of parallelism and non-uniform data access
  - Is suitable for multiple architectures, and multiple workloads.

    ⇒ Adaptability and resource-awareness are already included to some extend in X10

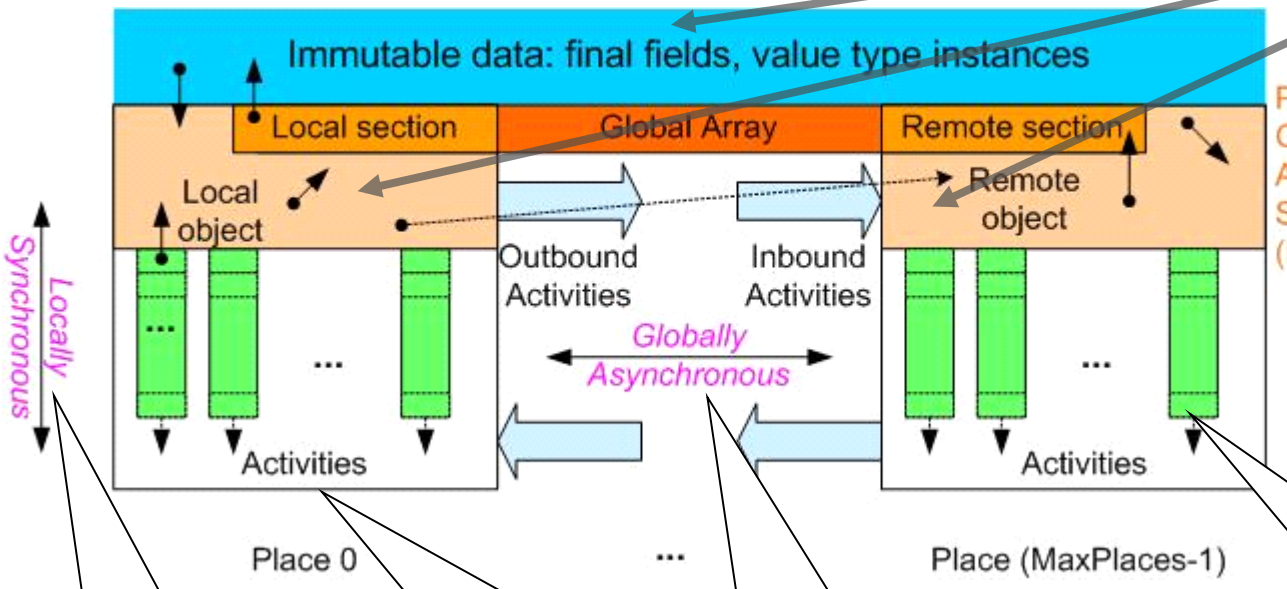
Note: Slides are partly based on material by Christoph von Praun, Vijay Saraswat, and Vivek Sarkar.

Goals of X10

- Simple
  - Start with a well-accepted programming model, build on strong technical foundations, add few core constructs
- Safe
  - Eliminate the possibility of errors by design, and through static checking
- Powerful
  - Permit easy expression of high-level idioms
  - And permit expression of high-performance programs
- Scalable
  - Support high-end computing with millions of concurrent tasks
- Universal
  - Present one core programming model to abstract from the current plethora of architectures

X10 Concepts

PGAS:
Replicated Data
Local Heap
Remote Heap



Locality Rule:
Any access to a mutable datum must be performed by a local activity
⇒ remote data accesses can be performed by creating remote activities

Place = collection of resident activities and objects

Activity = sequential computation that runs at a place

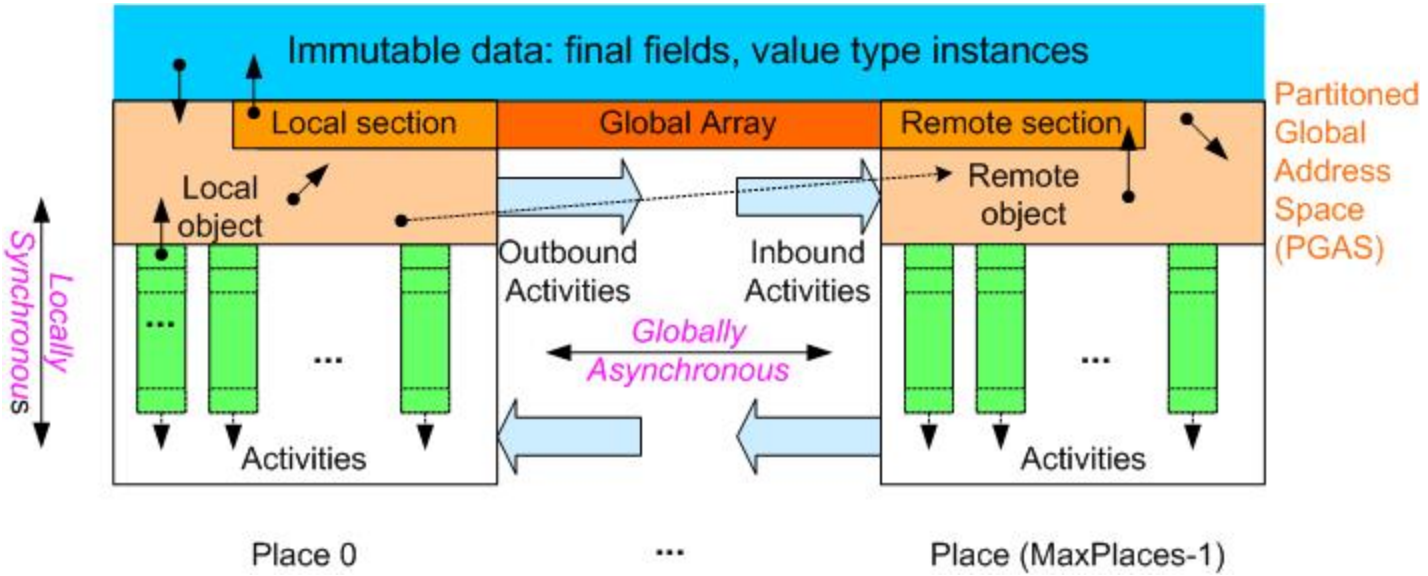
Locally Synchronous:
Guaranteed coherence for local heap
⇒ sequential consistency

Globally Asynchronous:
No ordering of inter-place activities
⇒ use explicit synchronization for coherence

Ordering Constraints (Memory Model)

Storage classes:
- Activity-local
- Place-local
- Partitioned global
- Immutable

X10 Constructs



Fine grained concurrency • `async` S	**Atomicity** • `atomic` S • `when` (c) S	**Global data-structures** •points, regions, distributions, arrays
Place-shifting operations • `at` (P) S	**Ordering** • `finish` S • `clock`	

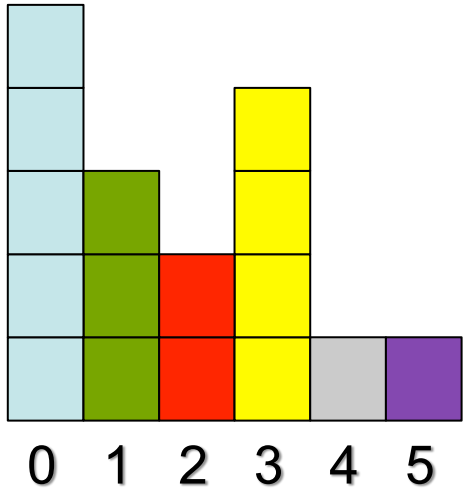
Two basic ideas: Places and Asynchrony

Invasive Histogram Computation

- Given: An array of 8 bit numbers
- Result: Number of observations in the array sorted into 256 bins

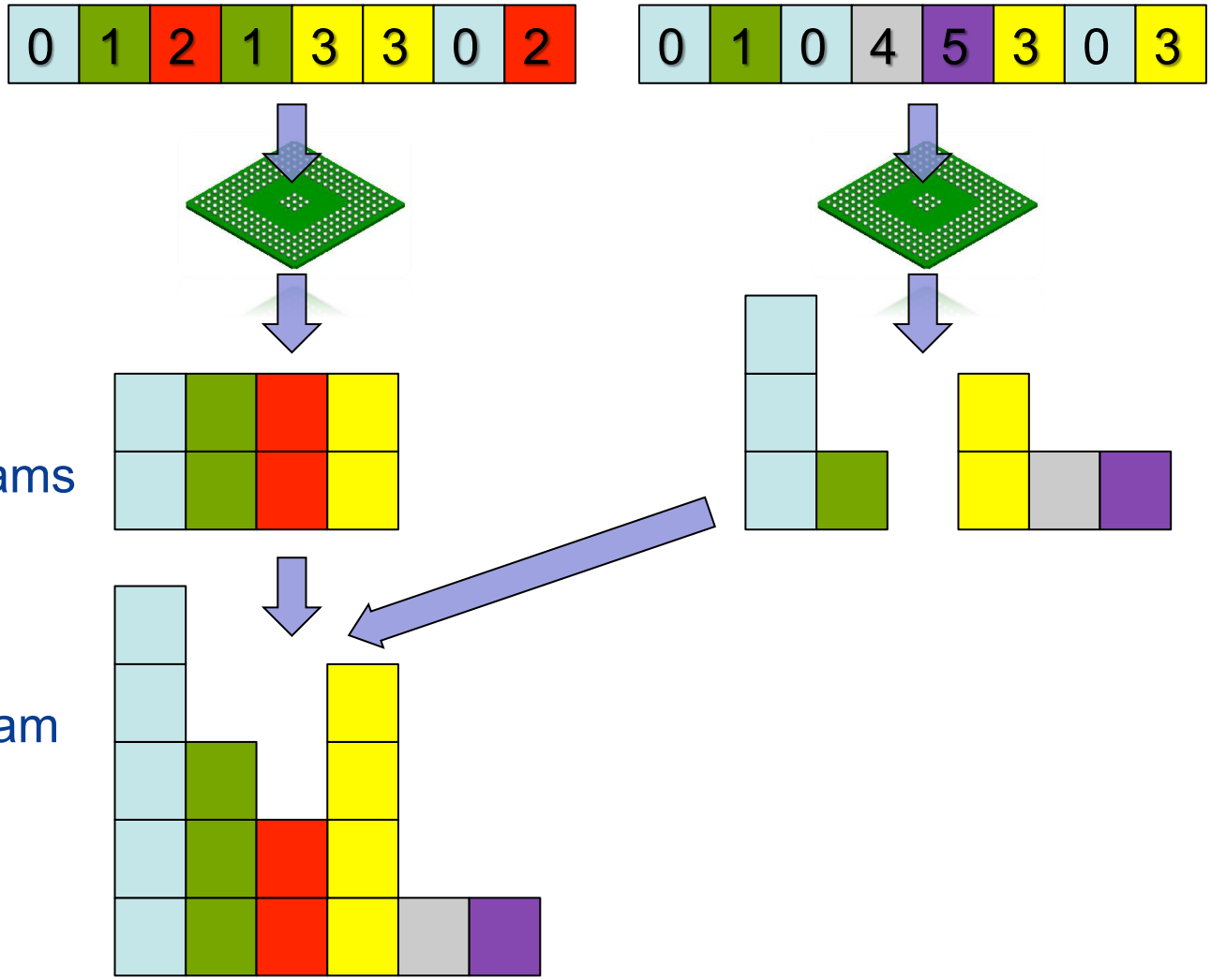
- Small illustrative example (array with 16 elements, 6 bins):

- Histogram:



Histogram, Parallelization

- 2 cores available:



- Sub histograms
- Histogram

Histogram, X10 Code

- Helper class for handling sub histograms

```
class subHist
{ val sh: Rail[Int] = Rail.make[Int](256, (Int) => 0);
  def this() {}

  def inc(i: Int)
  { sh(i)++;
  }

  def add(i: Int, v: Int)
  { sh(i) += v;
  }

  def get(i: Int): Int
  { return sh(i);
  }
  ...
}
```

Increments bin i by one

Generates 256 bins and initializes them with zero

Adds value v to bin i

Returns the value of bin i

Histogram, X10 Code

```
public class Histogram
{ public static def main(argv:Rail[String]!)
  { val rnd = new Random();
    rnd.setSeed(42);
    val noOfElements: Int = 10000000;
    val R = 0..noOfElements-1;  // Region
    val dataSet = new Array[Int](R, ((i):Point) => rnd.nextInt(256));

    val numberOfCores: Int = Invade();

    val D = Dist.makeDistribution(R, numberOfCores);
    val subHists: DistArray[subHist] =
        DistArray.make[subHist](Dist.makeDistribution(numberOfCores),
        ((i):Point) => new subHist());
    val distDataSet: DistArray[Int] =
        DistArray.make[Int](D, ((i):Point) => dataSet(i));
    ...
```

Generate some workload (random numbers within the interval [0,255])

Invade:
Are there some additional cores available?

Create mapping from region R to set of cores

Infect (Data):
Create at each core a sub histogram

Distribute workload to available cores

Histogram, X10 Code

```
...
finish
{ foreach (p in D.places())
  { async at(p)
    { for (i in D.get(p))
      { subHists(p.id()).inc(distDataSet(i));
      }
    }
  }
}

// Combine sequentially all sub histograms into a global one
for (i in 1..numberOfCores-1)
{
  ...
}
...
}
```

Infect (Computation):
Start at each place (core) the sub histogram computation asynchronously in parallel

`finish {S}`
Wait until all threads in **S** are finished, synchronization.
Retreat (Compute Resources)

Retreat (Memory Resources)

Notes

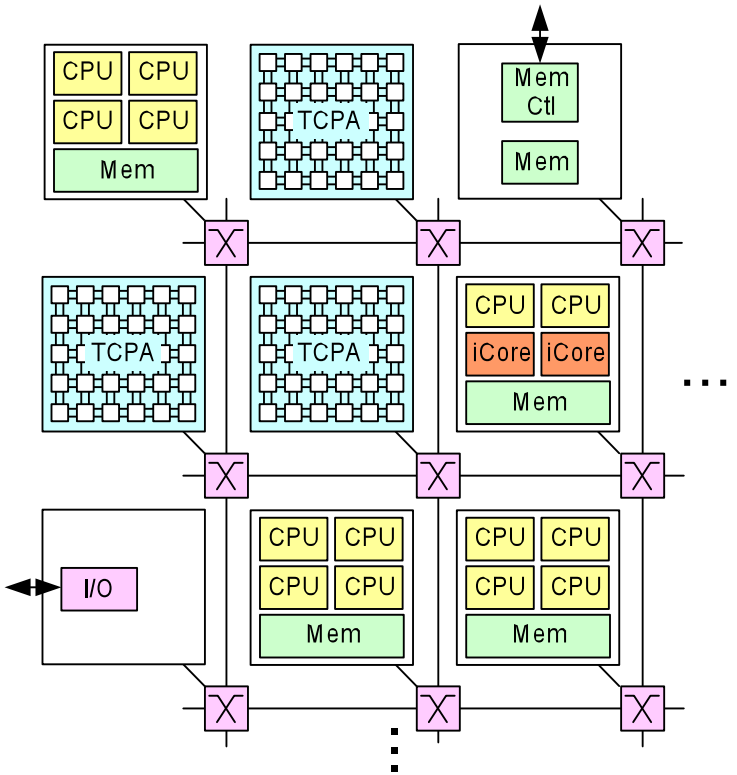
- The Invade function might have different flavors, e.g.,
  - Give me as much as possible resources
  - Give me a power of 2
  - Give me an even number
  - etc.

  - The type of the returned resource, (x86 multi-core, graphics device, ...) (CUDA devices are already supported to some extend in X10)

- According to the resource type, the program might select also different parallelization strategies, e.g.
  - Thread-level parallelism in case of a standard multi-core
  - Data-level parallelism in case of a graphics device

Project Area B – Architectures

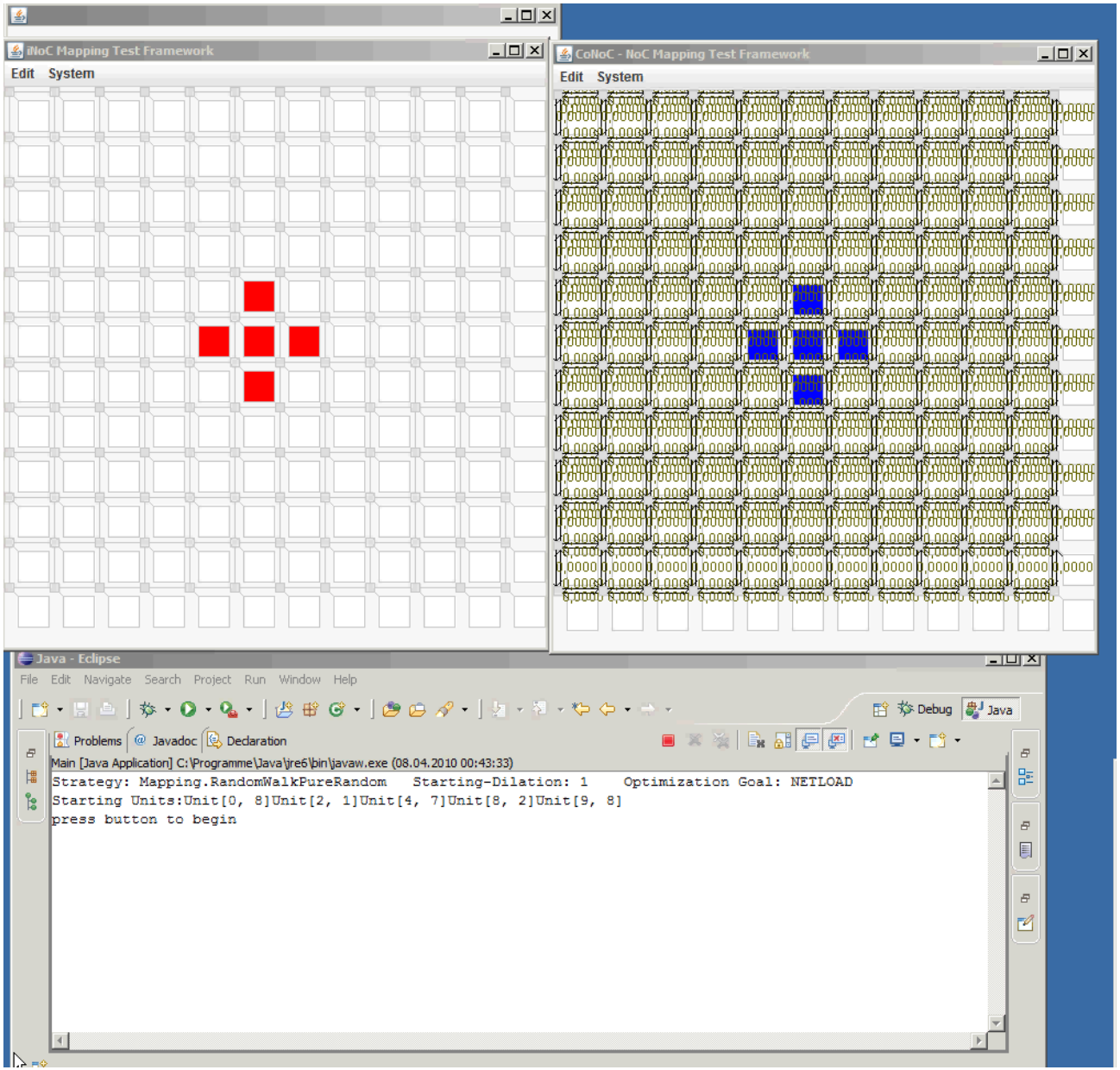
- Invasive Computer Architectures:
  - Invasion Control Architectures for networks of ASIP- (iCore [B1]), RISC- (CPU [B3]) and Tightly-Toupled Processor Arrays (TCPA [B2])
  - Microarchitecture:
    - Segmentable and reconfigurable memory, processor, instruction sets and interconnect [B1, B2, B4]
    - „Instruction set“ - definition for basic functionality [B1,B2]
    - Hardware-supported invasion (Invasion-Controller) [B2]
  - Macroarchitecture:
    - Hardware-supported Invasion (CIC [B3])
    - Invasive Communication Networks (iNoC [B5])
  - Monitoring and Design Optimization [B4]



Project Area C – Tools

- Run-Time System [C1]
  - Methods, principles and abstractions for extendable, (re-)configurable and adaptable OS structures for invasive computing systems
  - Agent technology for Scalable Resource Management
  - Techniques for Virtual Power Management
  - *i*RTSS: (de-centralized) Services of Operating Systems for Invasive Architectures
- Simulation and Compiler [C2, C3]
  - Simulation (Speed, HW/SW, Heterogeneity) [C2]
  - Compiler
    - Symbolic Parallelization: Loop Invader [C3]
    - Machine Markup Languages [C3]
    - Backend Design (X10 -> Sparc, X10 -> TCPA) [C3]
    - Invasification [C3]

Demo Invasion Strategies



Exercise 5

- Given a mesh-connected 2D invasive TCPA with NxN processor elements (PEs). Let the invasion of one single PE take one single clock cycle (time for one PE issueing an invasion request)
- CASE A:
  Let each PE be able to invade only one nearest neighbour PE after being invaded itself.
  - A) How many PEs may be invaded in n clock cycles?
  - B) Place an initial single PE seed program starting the invasion (Master of Invasion) such that a LINEAR INVASION (invasion in a single direction) yields the biggest CLAIM (number of invaded PEs).
  - C) How long does it take (minimally) to invade the whole NXN array assuming multi-D invasion is allowed?
    Specify a single invasion strategy.
- CASE B:
  Let each PE be able to once issue a simultaneous invade request to ALL its nearest neighbors in one clock cycle. Answer A),B),C) alike.
  (6 minutes)

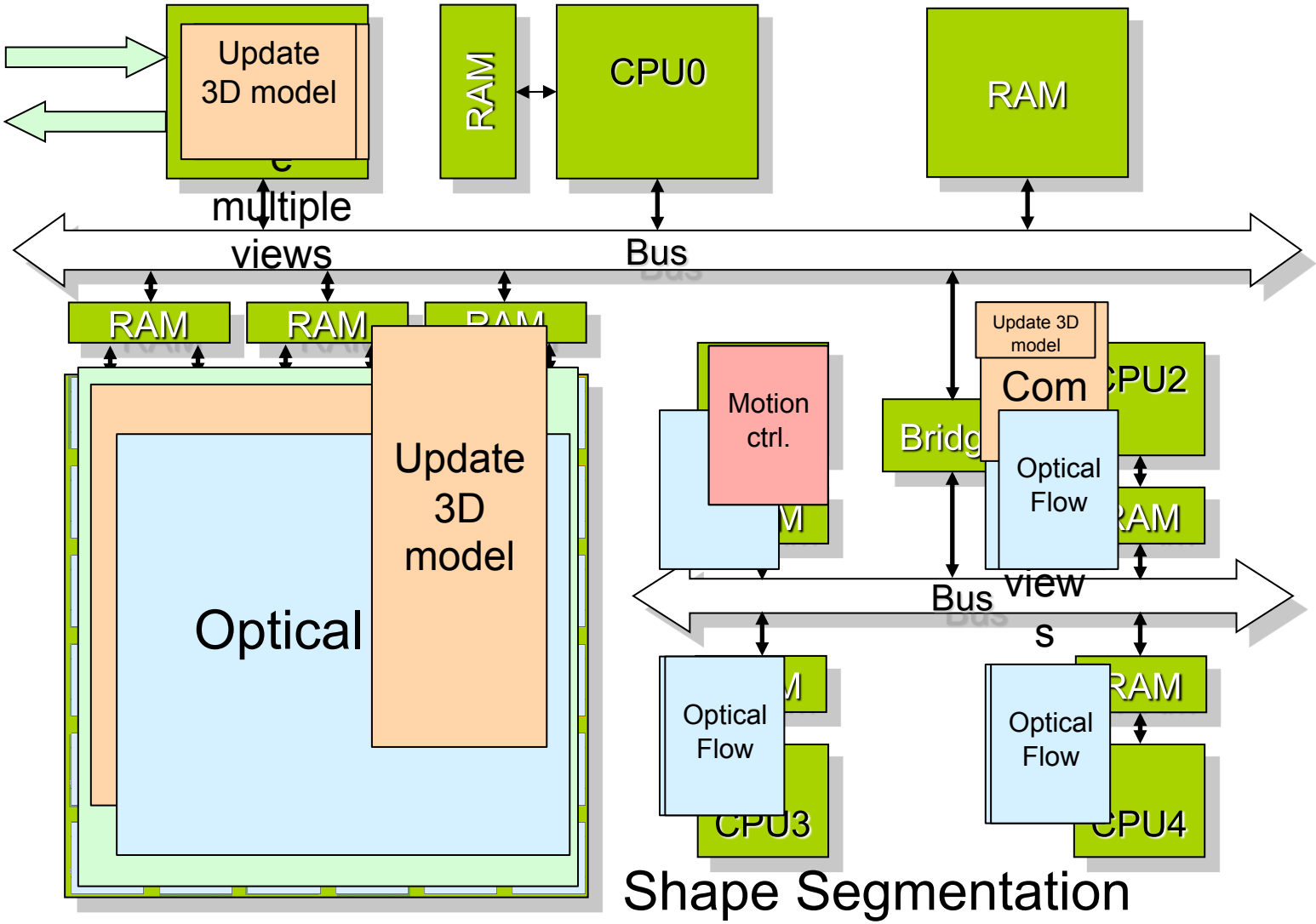
Project Area D – Applications

- Application Areas:
  - Robotics [D1]
    - Real-Time
    - Fault-Tolerance
    - Performance
  - Scientific Computing [D3]
    - Invasive Computing on HPC-Systems
    - Ressource utilization
    - Performance

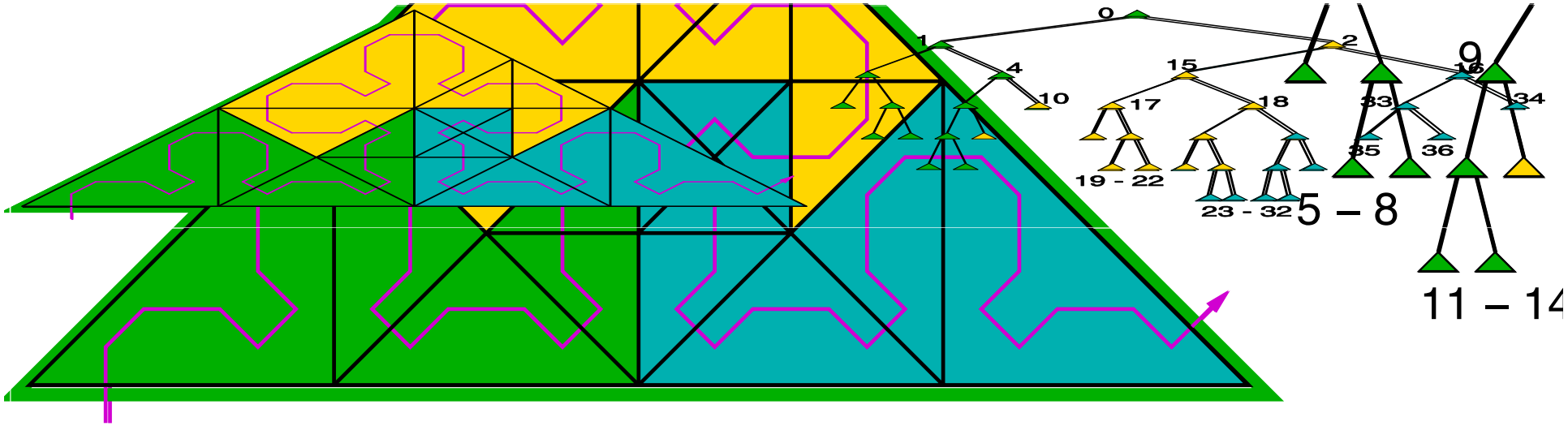
Invasive Computing Scenario in Robot Vision

Subtask	Sample algorithm	HW resources
Build 3D model of environment, use cameras from other robots	Stereo vision, disparity map, block matching, combine multiple camera views	video input, network I/O, TCPA, RISC
(Listen for “help” shouts, analyze audio input)		
Find regions of interest (ROI): persons, explosives	Color segmentation, region growing	TCPA
Check hypothesis for each ROI	Shape segmentation, shape matching	Multiple RISCs (one per ROI)
Motion planning	Shortest path calculation	1 RISC
Walk into hazardous area	Motion control	1 RISC hard real-time
Check for any interleaving objects	Optical flow	TCPA Multiple RISCs (e.g. 2)
Update 3D model	Disparity map update, combine multiple views	video input, network I/O, TCPA, RISC

Scenario Robot Vision



Scenario Multigrid Methods



Scenario: Dyn. Adaptive Mesh Refinement & „Multigrid“-Solver

- Structured mesh refinement based on tree structures
- Load distribution und -balancing using so-called „space-filling curves“ (strong changes in the dynamic mesh refinement, depending on application)
- Challenges: „Divide-and-conquer-“ as well as „Multi-level“-Approaches

Exercise VI

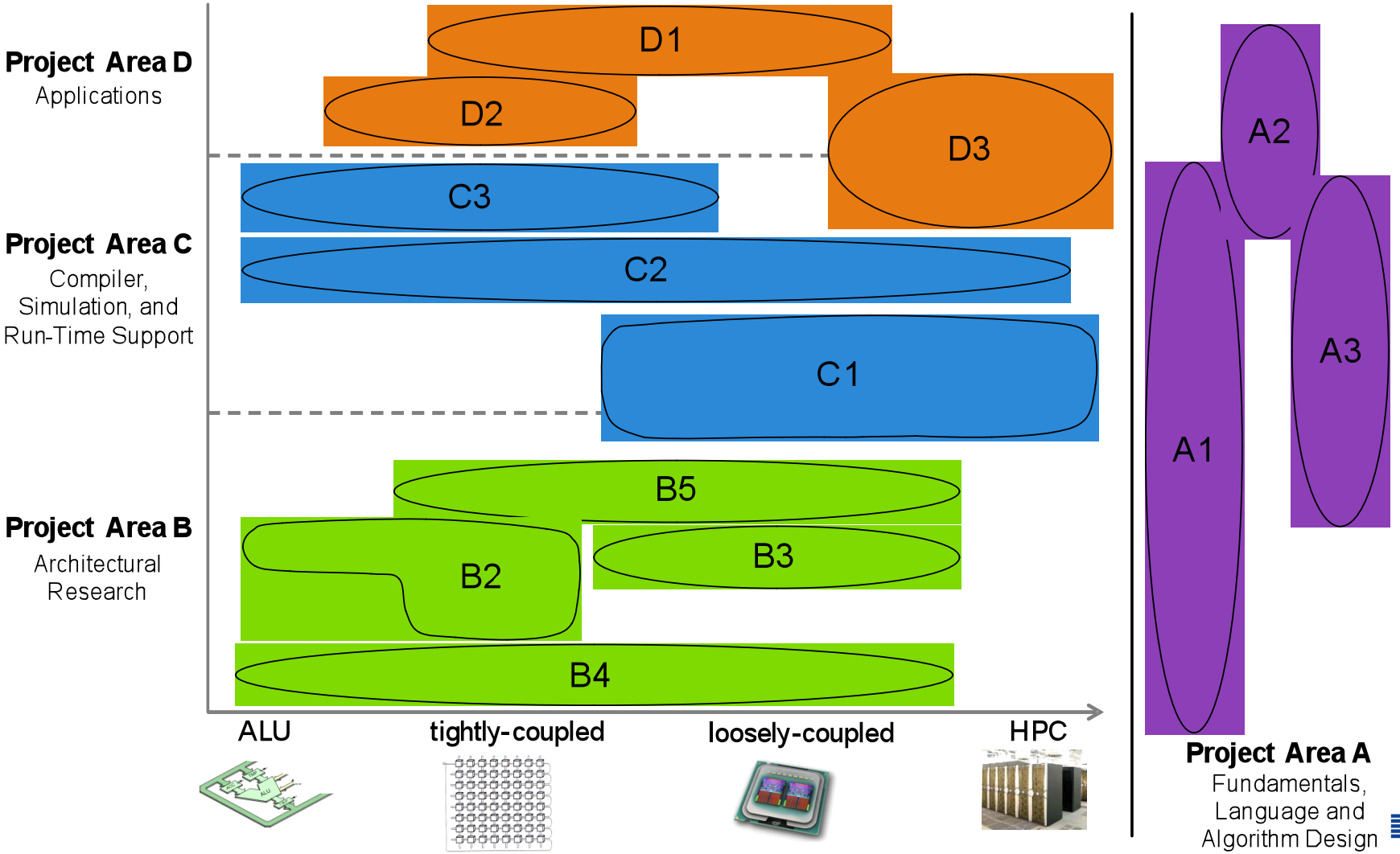
- A) Name applications and application areas where “INVASIVE COMPUTING” might beat conventional programming practice. In which aspects?
- B) For which applications/domains do you believe “INVASIVE COMPUTING” will be most beneficial? For which not. Explain.

App domain	Why?
Apps. with dynamic/time-variant degree of parallelism (DOP)?	…
Apps. with static parallelism?	…
Apps. with high/low number of threads?	…
Real-time apps.?	…

(4 minutes)

- What is Invasive Computing?
  - Uniquitousness of parallel computers
  - Challenges in the year 2020
  - Vision and Potentials
- Scientific Work Program
  - Basics: Resource-Aware Programming, Algorithms, Complexity
  - Architectures: Reconfigurability and Decentralized Resource Management
  - Tools: Compiler, Simulation Support and Run-Time System
  - Applications: Real-Time, Fault-tolerance, Efficiency and Utilization
- Structure, Chances and Goals
  - Project structure
  - Funded Institutions and Researchers
  - Demonstrator Roadmap
  - Impact and Risks

TRR 89 – Project Structure



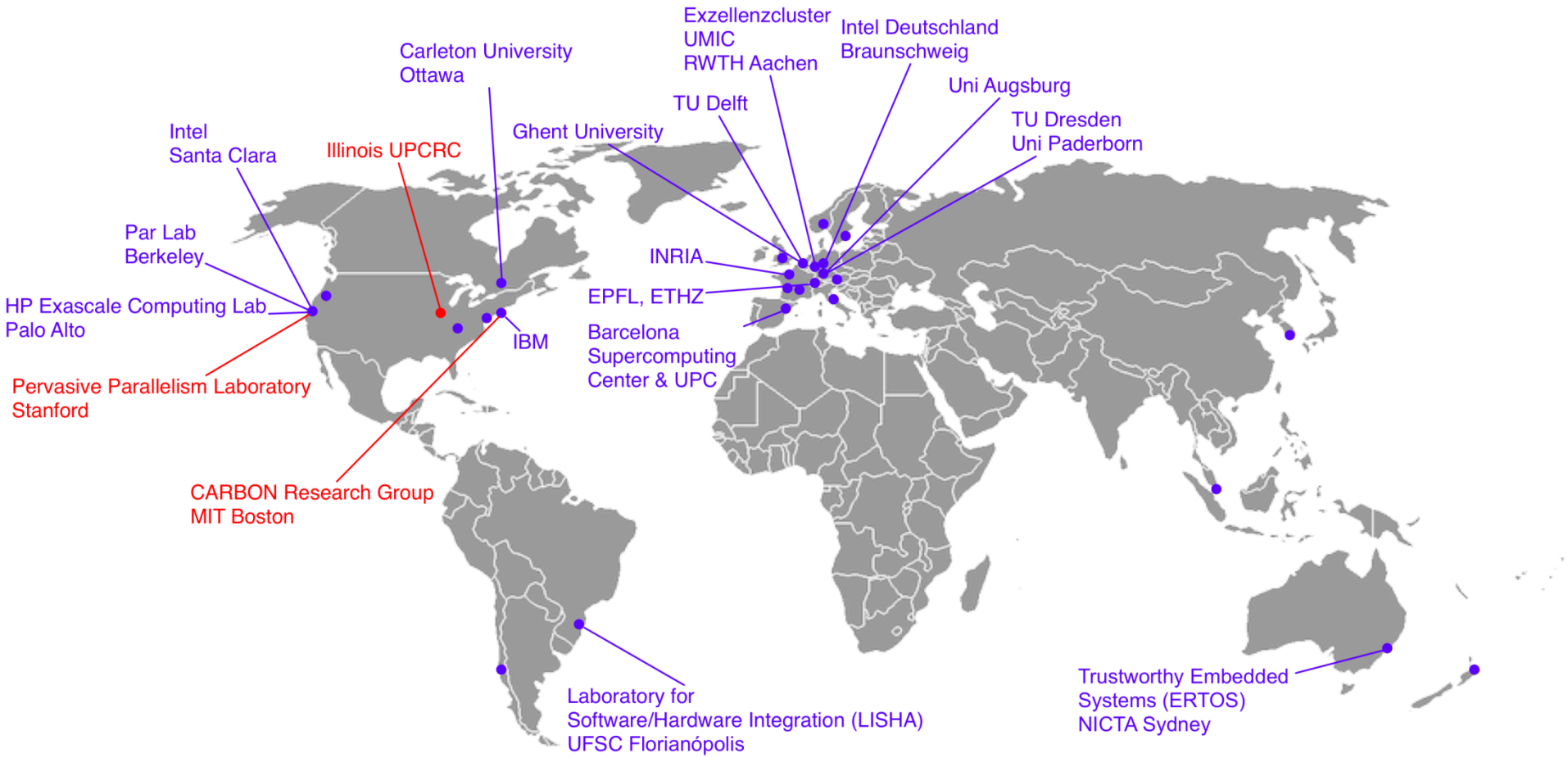
TRR 89 – Projects & Researchers

Project Area A: Fundamentals, Language and Algorithm Research	**Project Area B:** Architectural Research	**Project Area C:** Compiler, Simulation, and Run-Time Support	**Project Area D:** Applications
A1: Basics of Invasive Computing **Teich/Snelting**	**B1: Adaptive Application-Specific Invasive Micro-Architectures** **Henkel/Hübner/Bauer**	**C1: Invasive Run-Time Support System (iRTSS)** **Schröder-Preikschat/ Lohmann/Henkel/Bauer**	**D1: Invasive Software-Hardware Architectures for Robotics** **Dillmann/Asfour/ Stechele**
A2: Algorithms and Complexity of Invasion **Wanka**	**B2: Invasive Tightly-Coupled Processor Arrays** **Teich**	**C2: Simulation of Invasive Applications and Invasive Architectures** **Hannig/Gerndt/Herkersdorf**	**D2: Invasive Ray Tracing** **Stamminger**
A3: Scheduling and Load Balancing **Sanders**	**B3: Invasive Loosely-Coupled MPSoC** **Herkersdorf/Henkel**	**C3: Compilation and Code Generation for Invasive Programs** **Snelting/Teich**	**D3: Multilevel Approaches and Adaptivity in Scientific Computing** **Bungartz/Gerndt**
	B4: Hardware Monitoring System and Design Optimization for Invasive Architectures **Schmitt-Landsiedel/Schlichtmann**		
	B5: Invasive NoCs **Becker/Herkersdorf/Teich**		

TRR 89 – Existing Cooperations

- Successful cooperations exist with famous Chip and System Design Houses, e.g.,
  - Infineon, Munich
  - Alcatel-Lucent, Neremberg
  - IBM, Böblingen & Poughkeepsie
  - Cadence, Munich
  - Forte Design Systems, San Jose
  - Intel GmbH, Germany

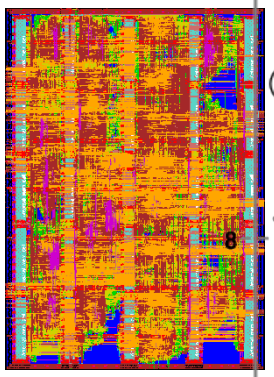
TRR 89 – International Initiatives & Contacts

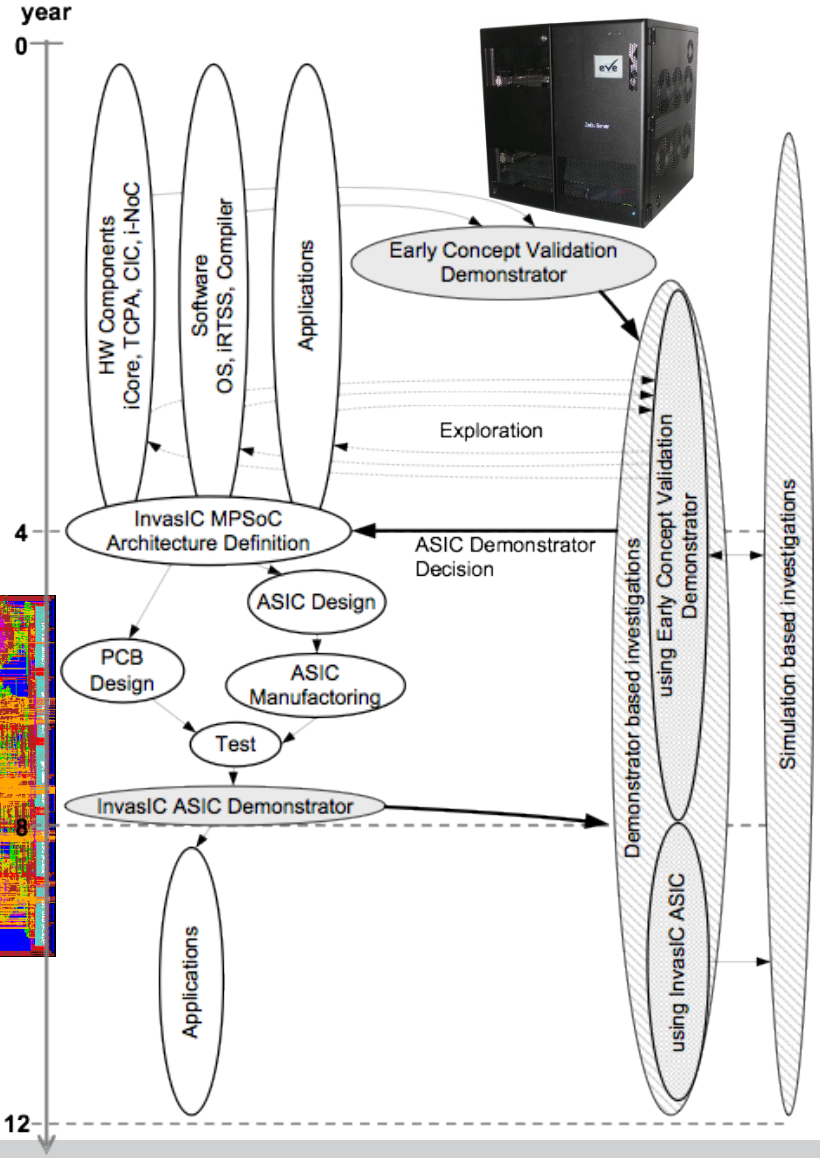


Memberships in European Networks of Excellence:

TRR 89 – Validation & Demonstrator Roadmap

- 2-level validation concept:
  - Phase I:
    Early Concept Validation Demonstrator (FPGA-based)
  - Phase II:
    InvasIC ASIC Demonstrator
- InvasIC Lab (TP Z2)
  - Each location one lab
  - 1 technician per site
  - Established milestone roadmap





Impact and Risks

- Introduction of a new paradigm of resource-aware programming as well as new architectural support by reconfigurable MPSoC-Architectures: **InvasICs**

- Expected impact on:
  - Future advanced processor development for MPSoCs
  - Future programming environments for Many Core Systems
  - Development of parallel algorithms

- Potential Risks:
  - Acceptance of resource-aware programming
  - Cost of Invasion (Hardware/Software, Timing)

Questions?