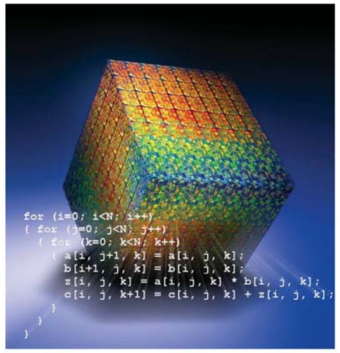
Invasion: Application-Driven Resource Management for Future MPSoCs

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG



```
for (i=0; i<N; i++)
{ for (j=0; j<N; j++)
  { for (k=0; k<N; k++)
    { a[i, j+1, k] = a[i, j, k];
      b[i+1, j, k] = b[i, j, k];
      z[i, j, k] = a[i, j, k] * b[i, j, k];
      c[i, j, k+1] = c[i, j, k] + z[i, j, k];
    }
  }
}
```

Karlsruhe Institute of Technology

TRR 89

J. Teich, 12th OC Colloquium, Nuremberg, 15. September 2011

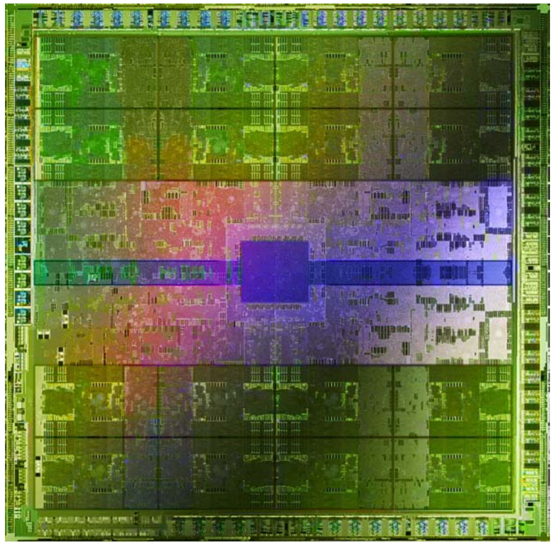
Outline

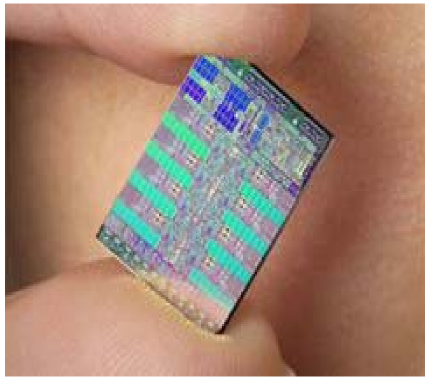
- What is Invasive Computing?
  - Uniquitousness of parallel computers
  - Challenges in the year 2020
  - Vision and Potentials

- Scientific Work Program
  - Basics: Resource-Aware Programming, Algorithms, Complexity
  - Architectures: Reconfigurability and Decentralized Resource Management
  - Tools: Compiler, Simulation Support and Run-Time System
  - Applications: Robotics, Scientific Computing

- Structure, Chances and Goals
  - Project structure
  - Funded Institutions and Researchers
  - Demonstrator Roadmap
  - Impact and Risks

Ubiquitousness of parallel computers



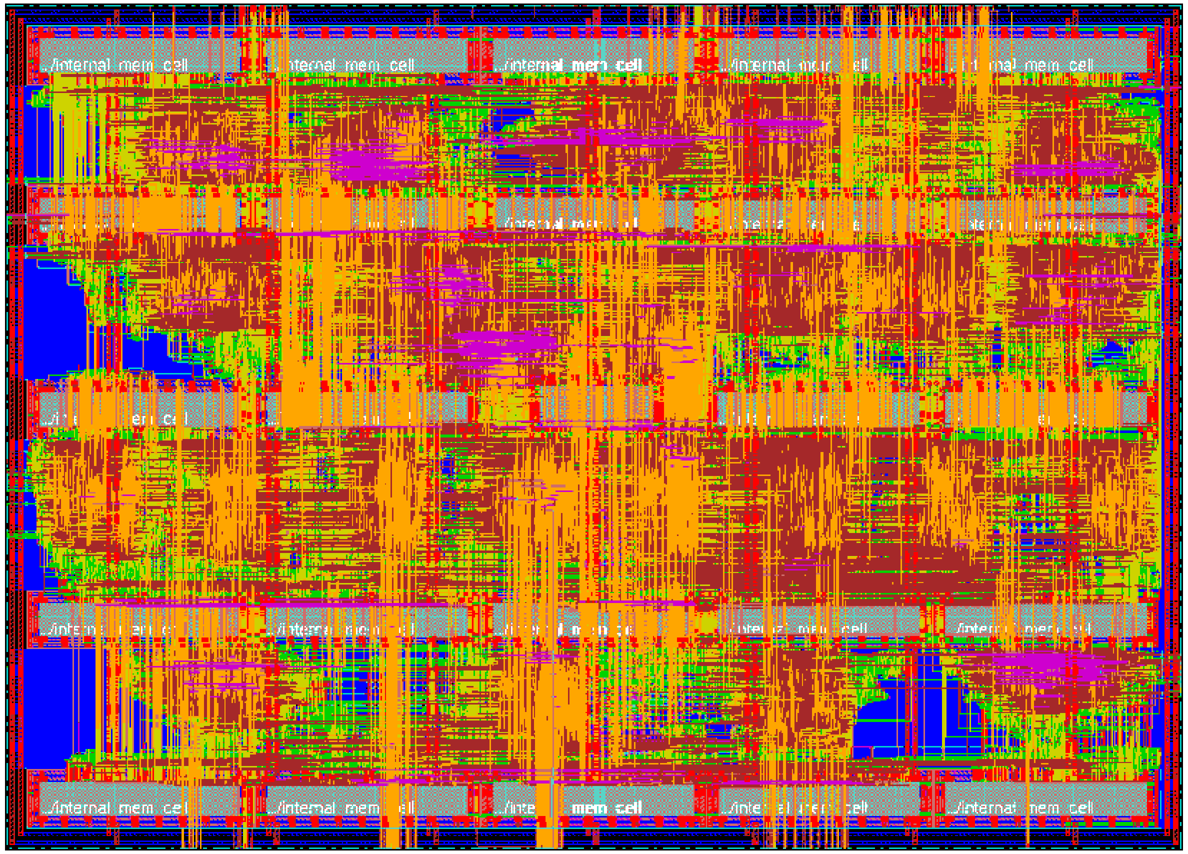


Nvidia Fermi: 512 Cores

Sony Playstation 3, IBM Cell 9 Cores

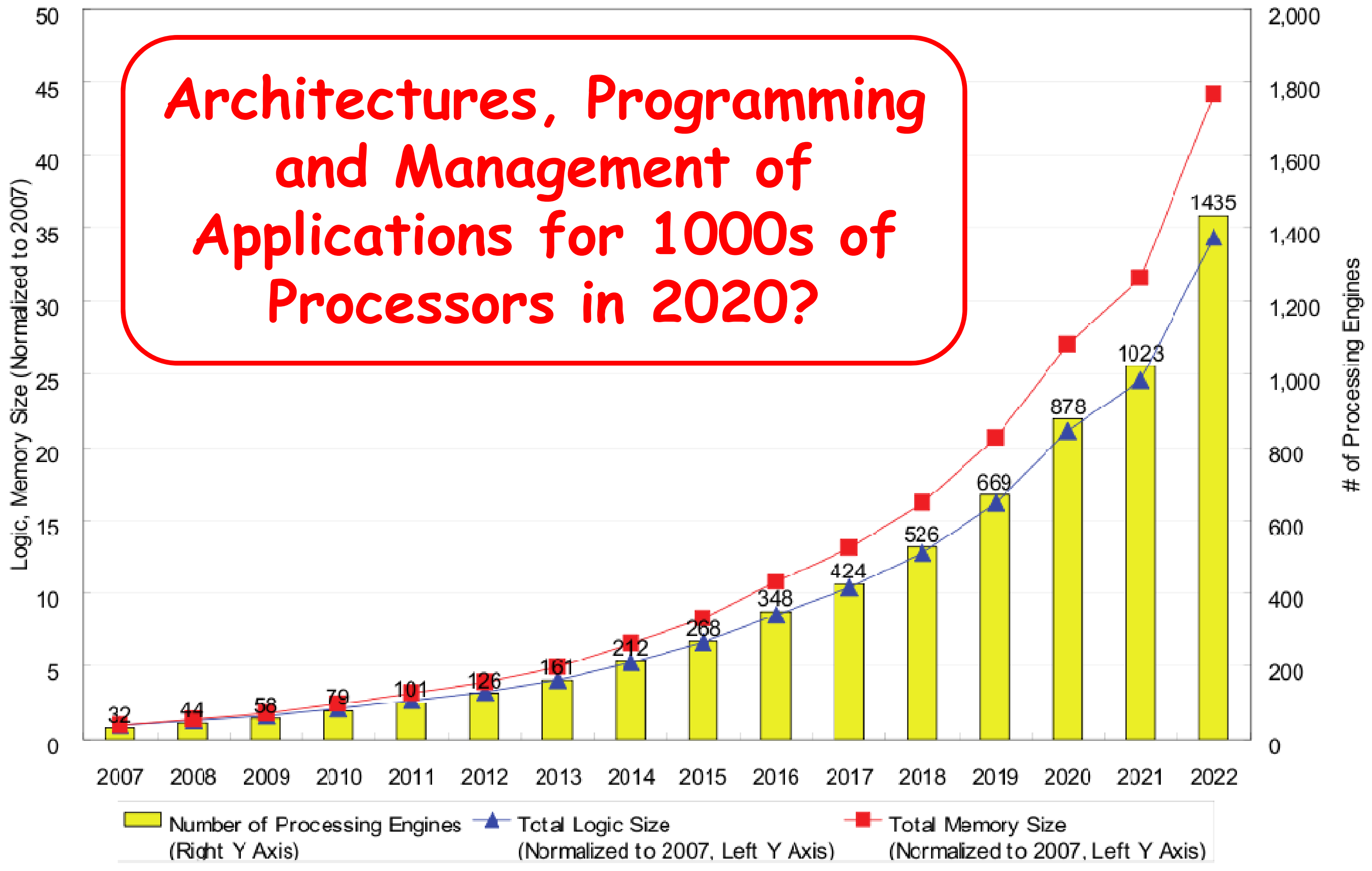
Intel SCC: 48 cores

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.

Challenges in the year 2020

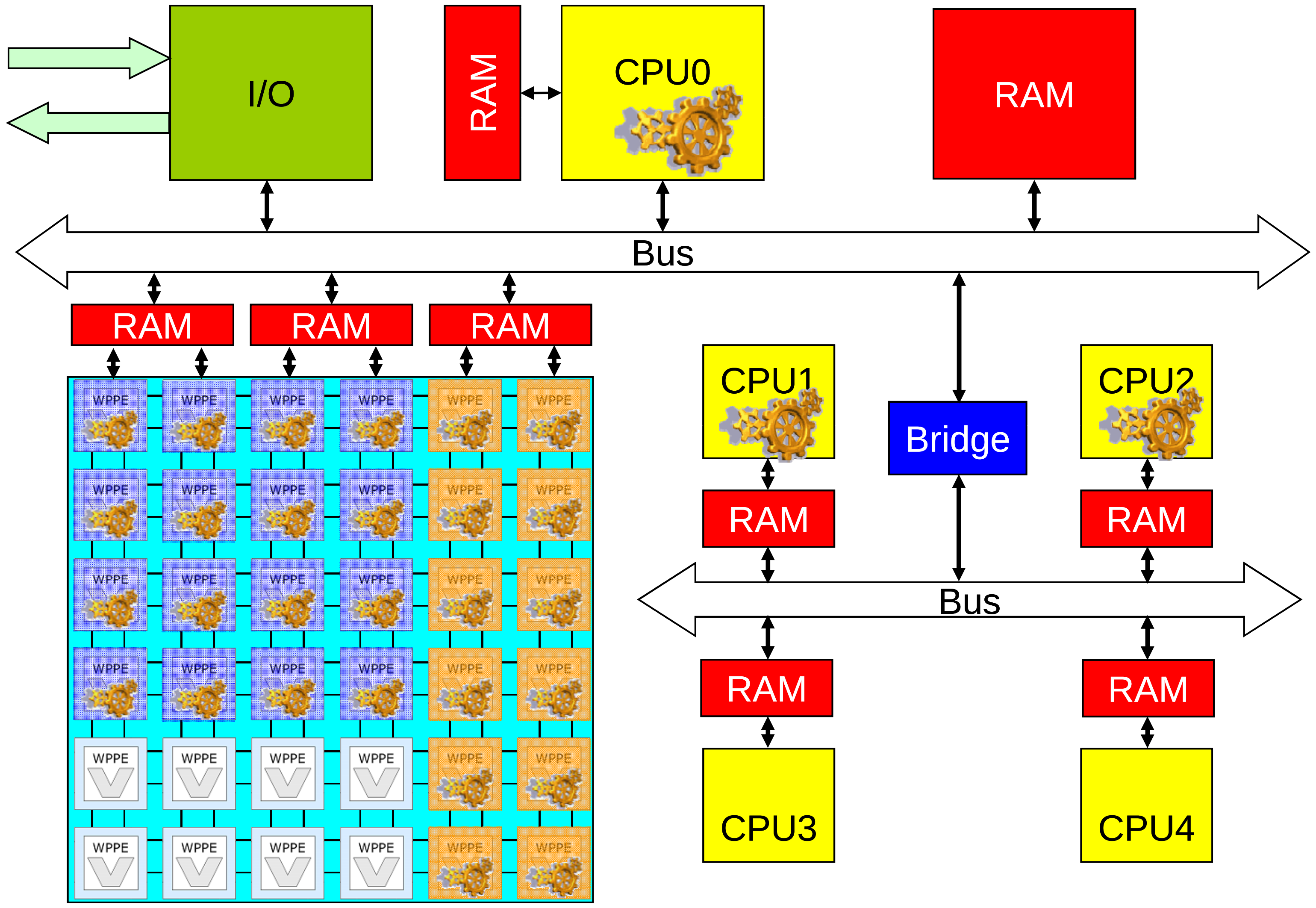


SOC Consumer Portable Design Complexity Trends

Challenges in the year 2020

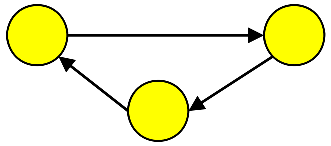
- Complexity
  - How to map dynamically applications onto 1000 or more processors while considering memory, communication and computing resource constraints?
- Adaptivity
  - How and to what degree shall algorithms and architectures be adaptable (HW/SW, bit/word/loop/thread/process-level)?
- Scalability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

Invasion: Example



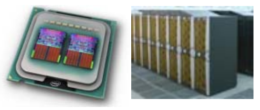
Considered Abstraction Levels

process-level, thread-level



Hw + Sw Control

Multi Core

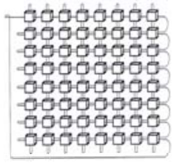


loop-level

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

Hw-Ctrl.+ Func.

Processor Array

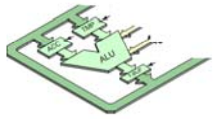


instruction-level

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

Hw-Ctrl. / VLIW

FUs



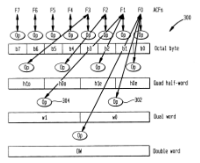
word-level, bit-level

0101000110101010101010

1010101010001111111

Hw-Ctrl. / VLIW

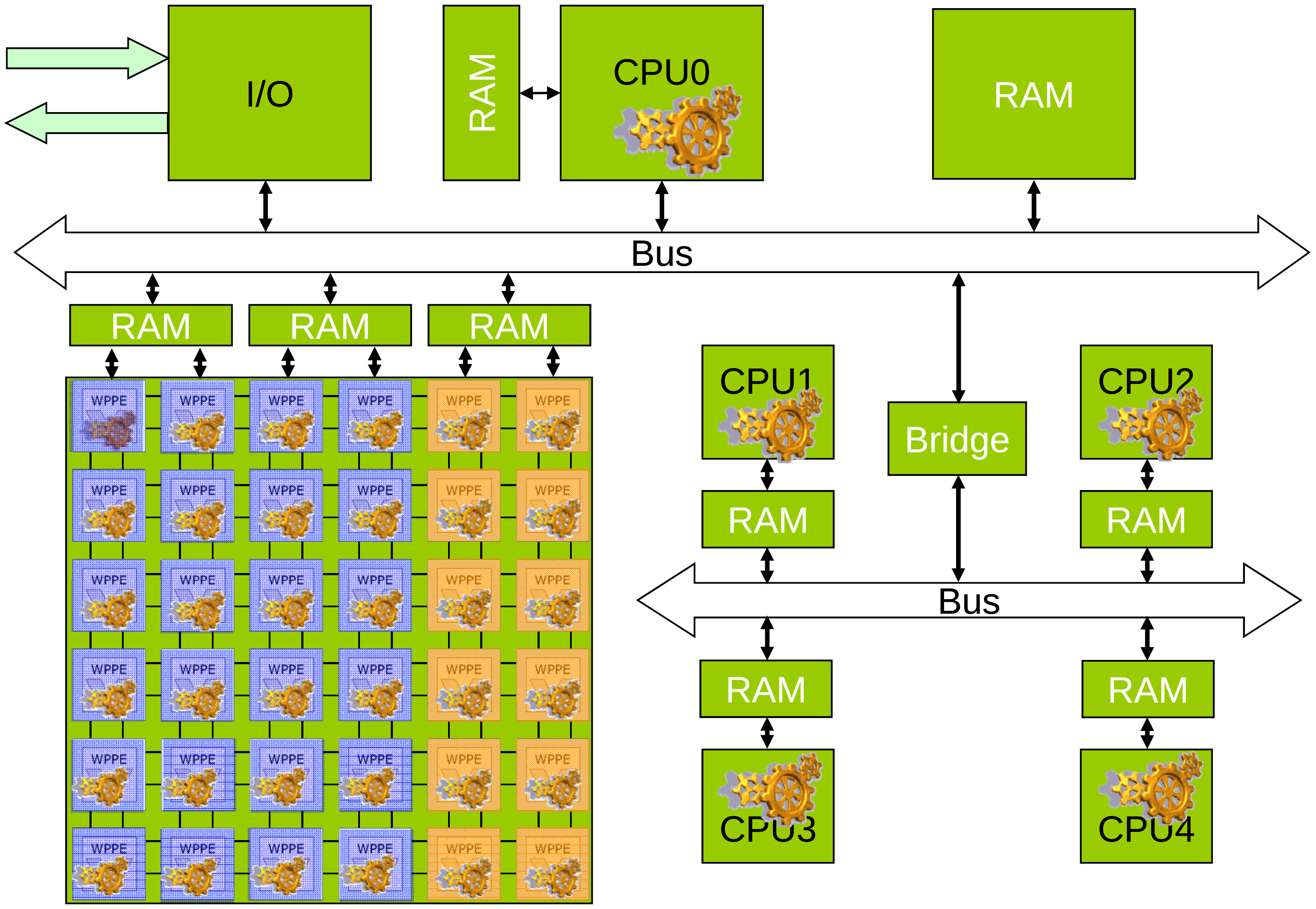
SW-Units



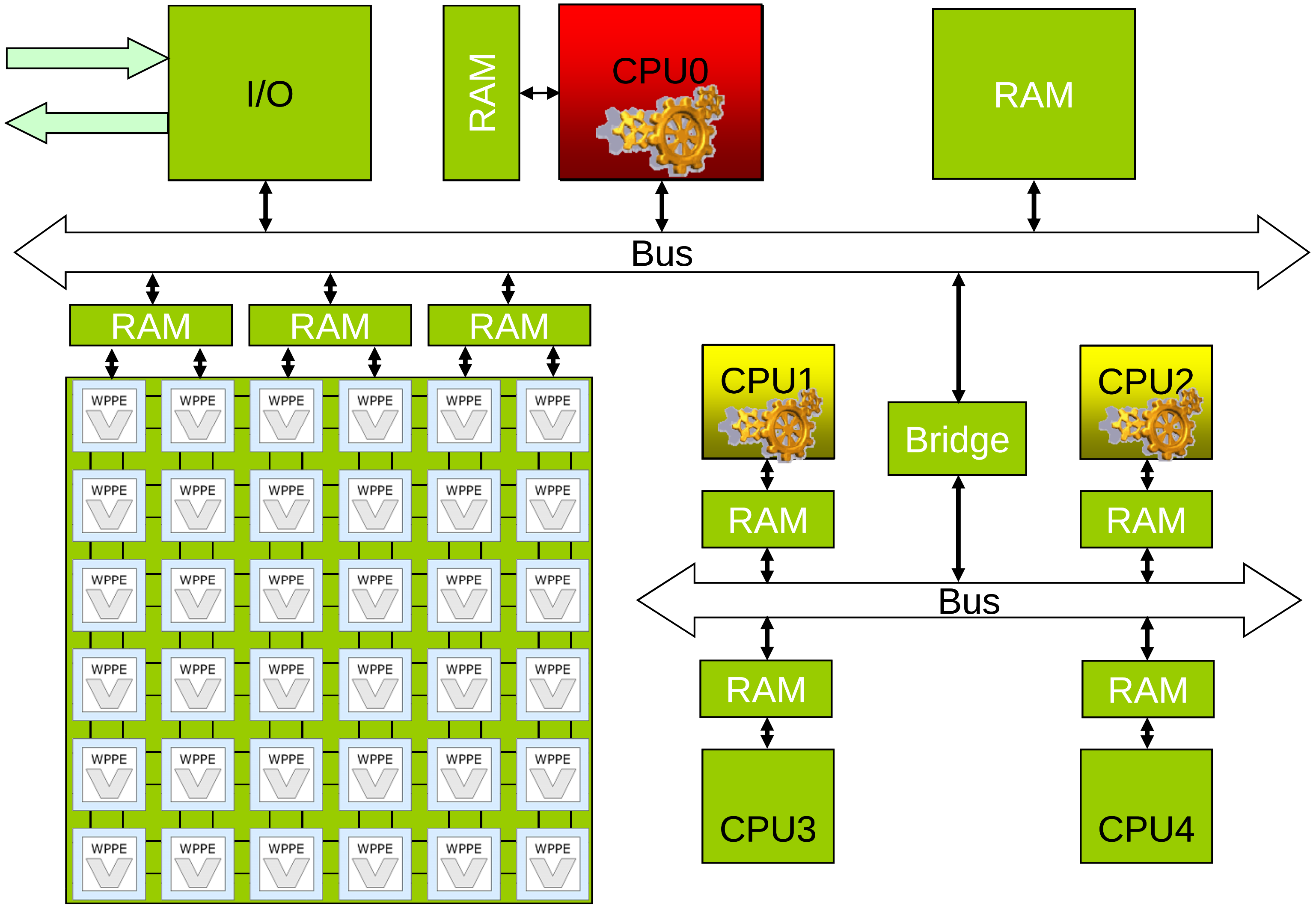
Vision and Potentials

- 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.
  - Resource 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

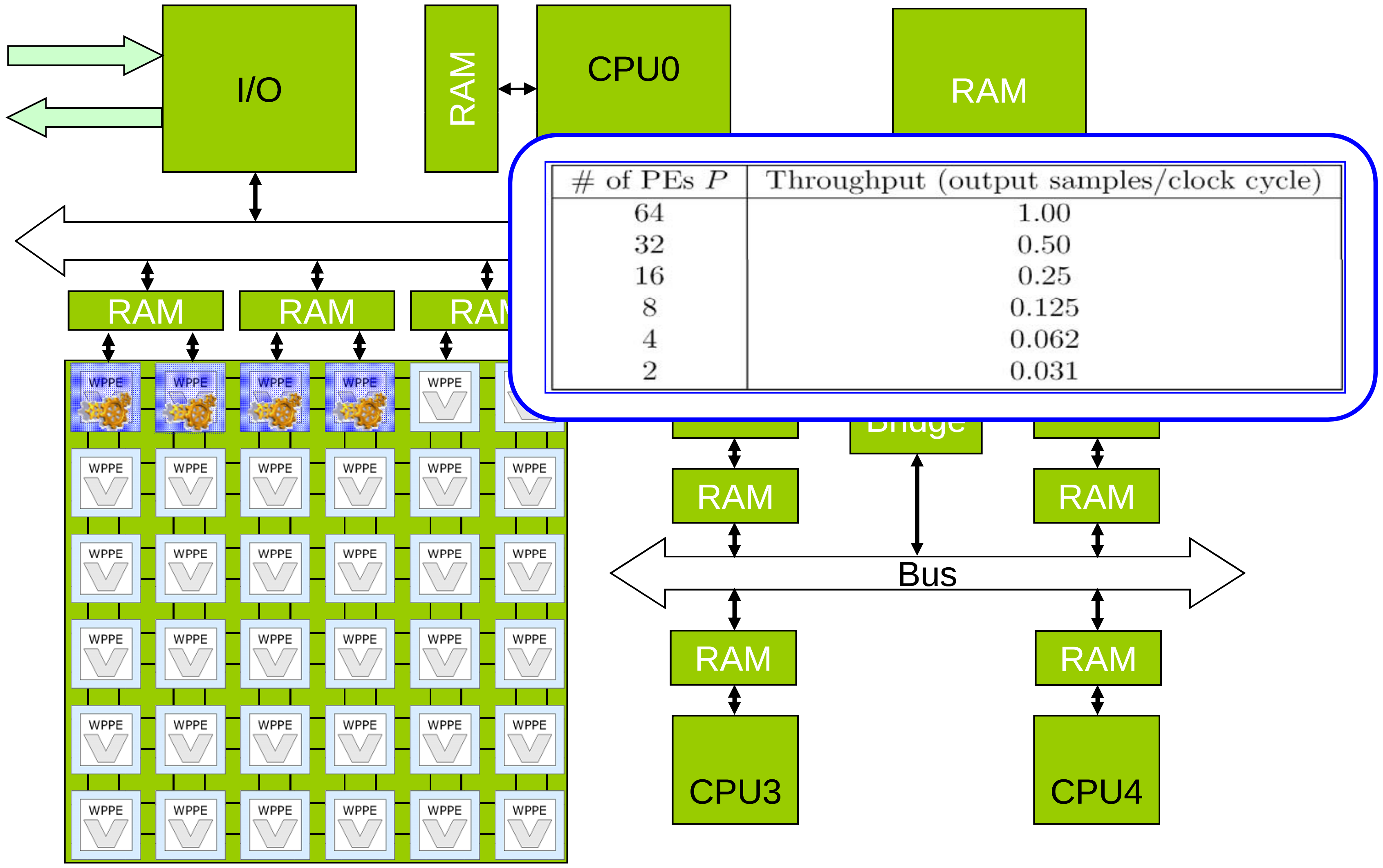
Potential: Resource Utilizations up to 100%



Potential: Power and Temp. Management

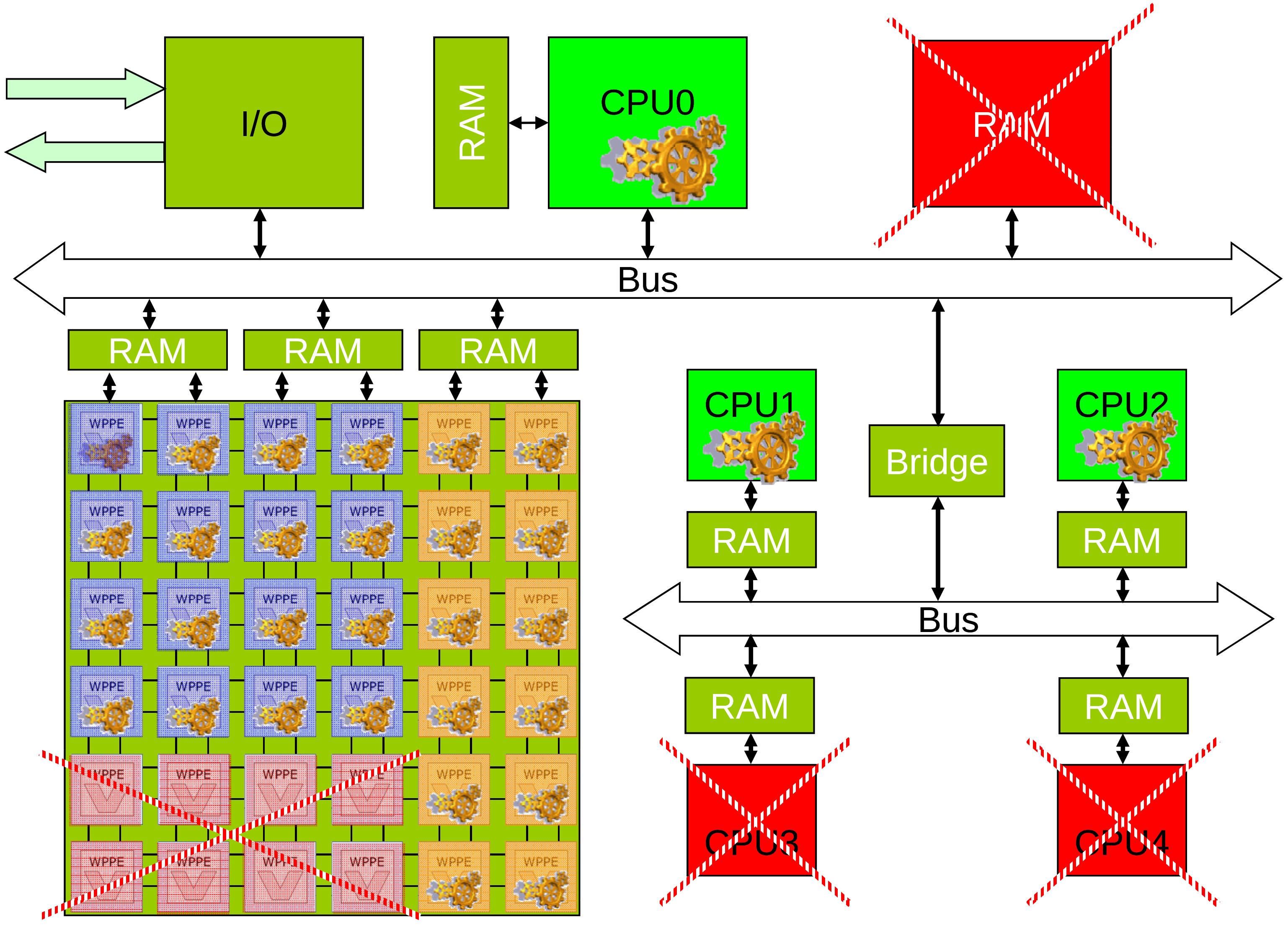


Potential: Performance Gain/Tradeoff



# 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



Outline

- 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: Robotics, Scientific Computing

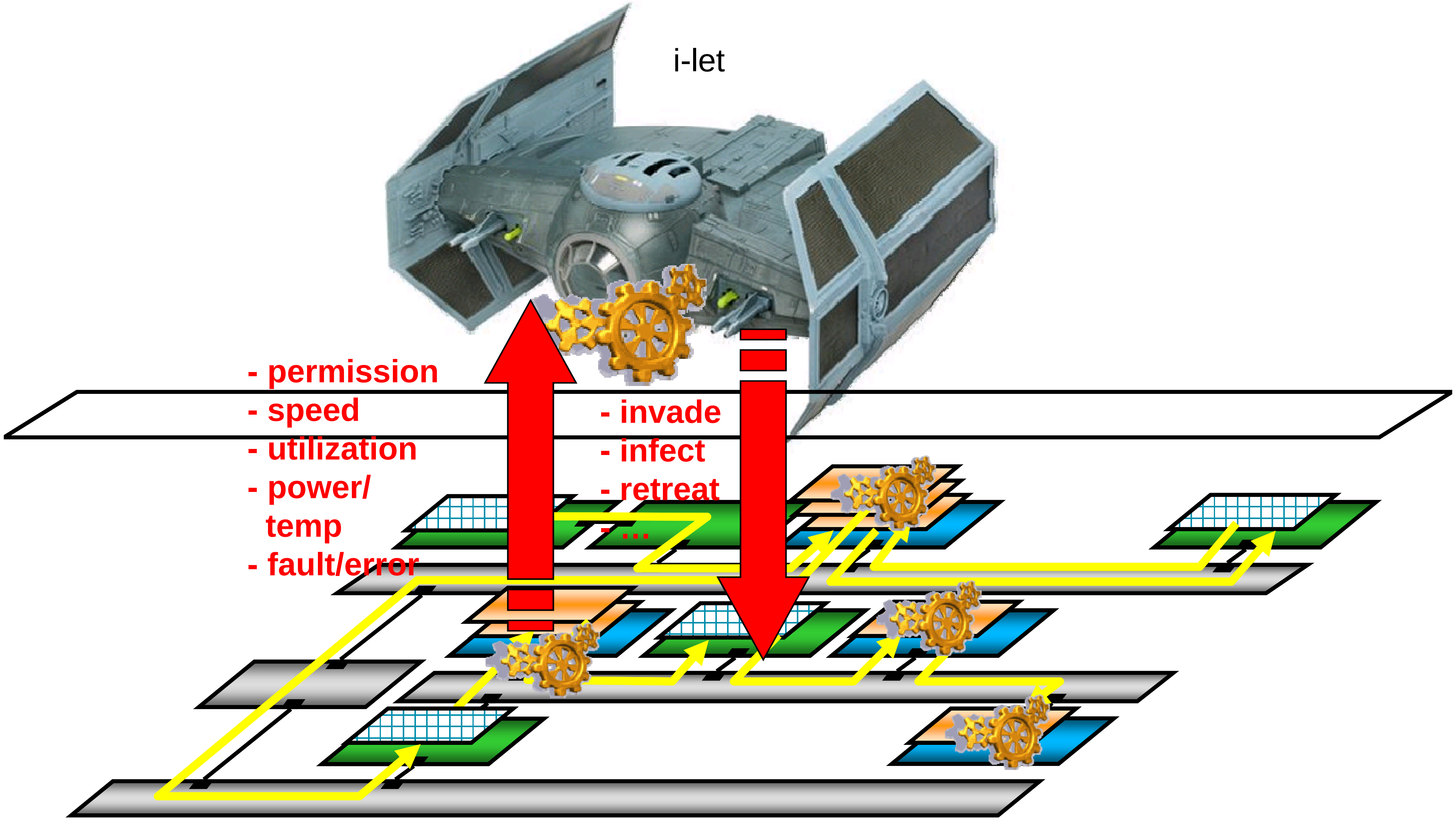
- Structure, Chances and Goals
  - Project structure
  - Funded Institutions and Researchers
  - Demonstrator Roadmap
  - Impact and Risks

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



Project Area A – Basics

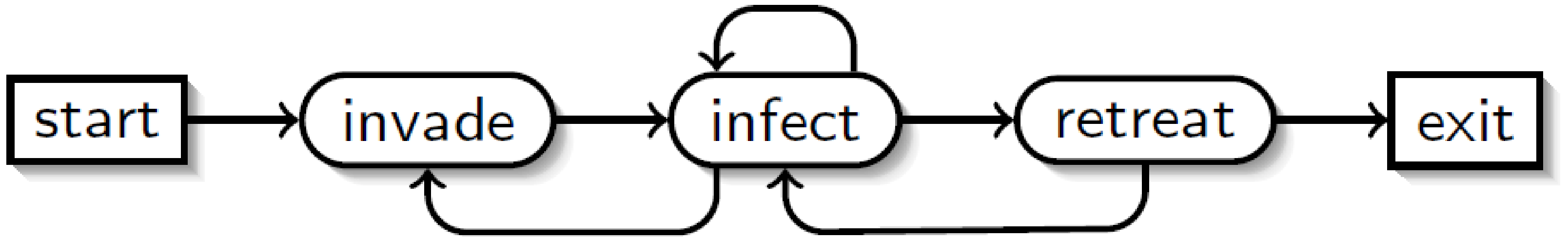
- Programming and Language Issues:
  - Finding and classification of elementary (basic) constructs for invasive programs (the *invasive command space*) [A1]
  - Definition of an abstract kernel language (syntax, semantics, type system) [A1]
  - Embedding of command set into programming language(s) [A1]

- Mathematical Models for Effifiency and Utilization Analysis of invasive applications [A1]

- Algorithm Engineering:
  - Complexity and cost invasive algorithms [A1]
  - Scheduling and Load Balancing [A3]

Basic Invasive Programming Constructs

- **Invade**
  - Allocation and reservation of system resources
    - Processors
    - Communication channels
    - Memory
  - Returns a claim (allocated resources)
  - Depends on the applications demand of parallelism
  - Depends on the current state of the resources (resource-aware)
- **Infect**
  - Copying program code and data to the claimed resources
  - Parallel execution of the program i-lets (code + data)
- **Retreat**
  - Frees occupied resources



Invasive Programming Constructs

- **Definitions**
  - i-let:
    - A piece of a program for invasive-parallel execution (code+data)
  - claim:
    - Set of allocated resources
      (processors, memory, communication)
- **Realization**
  - Using existing parallel programming languages, instead of designing a new language
  - Decision: Extension of X10 programming language
  - Using X10 as base language for invasive computing
  - Library-based approach

Languages and APIs for Parallel Programming

	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	(✓)	✓	✓	✓		✓		✓		✓		✓		✓	(✓)	✓

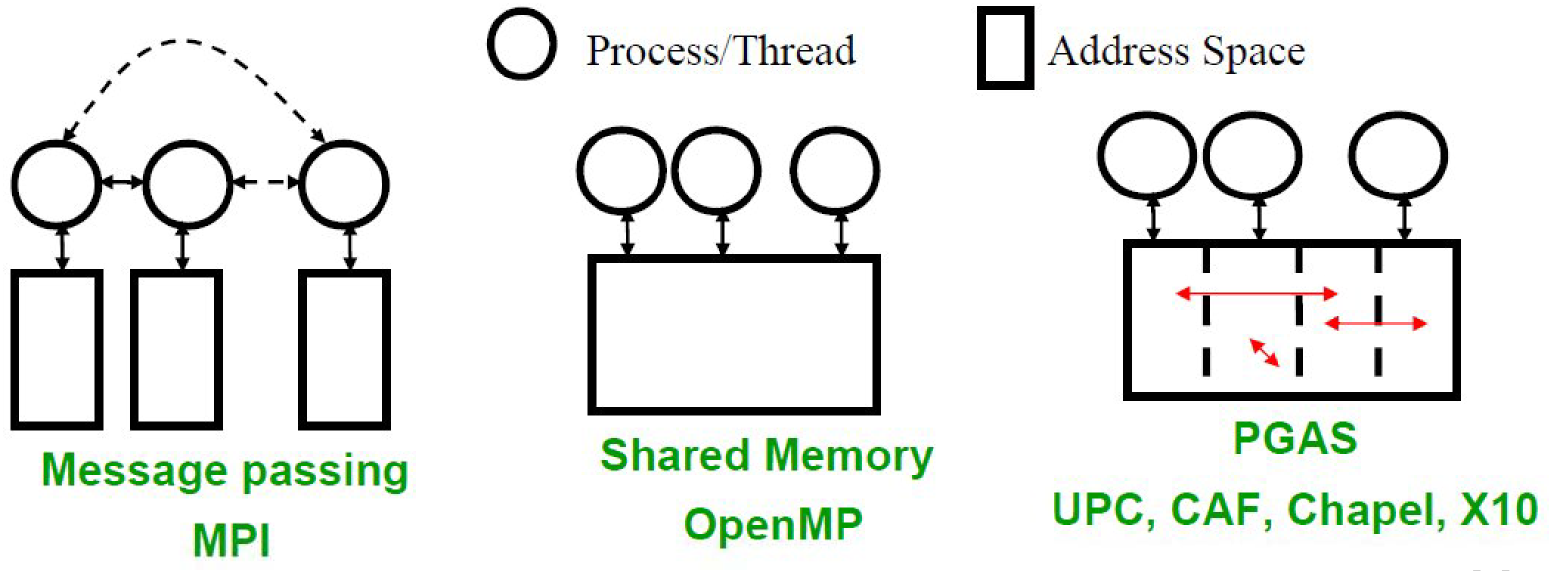
Outline

- Invasive Programming in X10
  - Introduction to X10
  - Invasive Programming Library
- Simulation of Invasive Programs and MPSoC architectures
  - Goals
  - Simulation Model
  - Case Study
  - Future Work

X10 Programming Language

- **X10 Programming Language**
  - Parallel, object-oriented programming language
  - Developed by IBM (since 2004)
- **General Properties**
  - Supports distributed, heterogeneous processor and memory architectures
  - Syntax between Java and Scala
  - OO language features:
    - Classes, objects, inheritance, generic types
  - Functional language features:
    - Type inference, anonymus functions, closures, pattern matching
  - Parallel constructs:
    - Concurrency, synchronization, distribution, atomicity
  - PGAS Programming Model

PGAS Programming Model



Source: [1]

- **PGAS: Partitioned Global Address Space**
  - Threads of a program have a global view, they share the same address space
    - Each thread sees the entire data set
    - No need for replication of data, as in the case of message passing
  - Address space is divided into partitions
    - Partitions may be physically distributed
    - Threads may reference data at other partitions (remote references)
    - Programmer is aware of data sharing among threads

X10 Parallel Constructs

- **PGAS memory is called Place in X10**
- **PGAS thread is called Activity in X10**

- **Activity**
  - Light-weight thread (user-level, not POSIX)
  - Creation with async
  - Synchronization via finish, atomic
  - Activities cannot be named or aborted

- **Place**
  - Notion of a shared memory multi-processor
  - Potentially different compute capabilities
  - Holds activities and objects
  - New places cannot be created at runtime

`async {S}`

- Creates a new child activity at the current place and asynchronously executes S
- Returns immediately

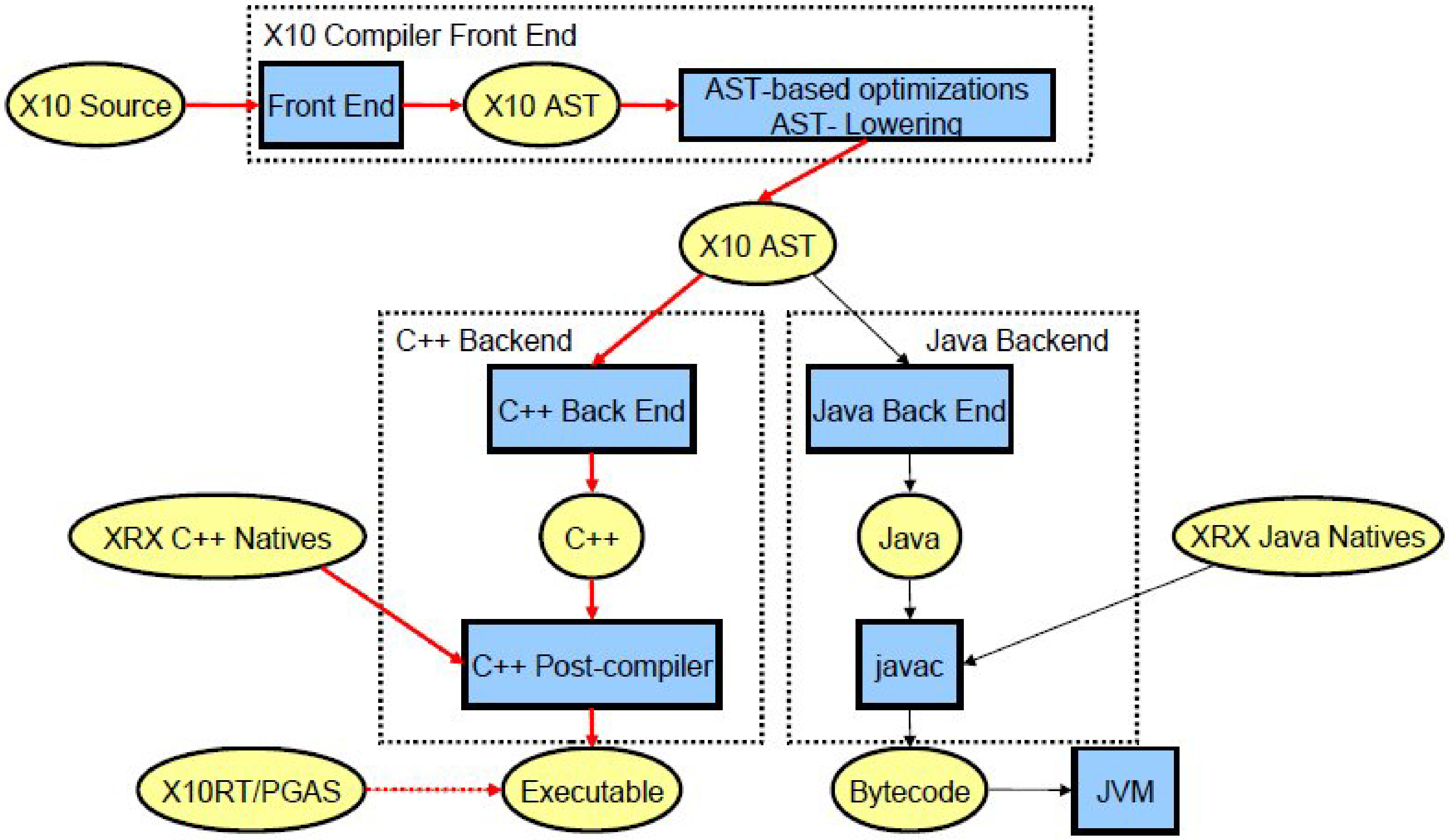
`finish {S}`

- Executes S and waits until all recursively spawned activities are finished

`at (P) {S}`

- Executes S at place P
- Current activity blocks
- Copy semantics

X10 Compile Flow



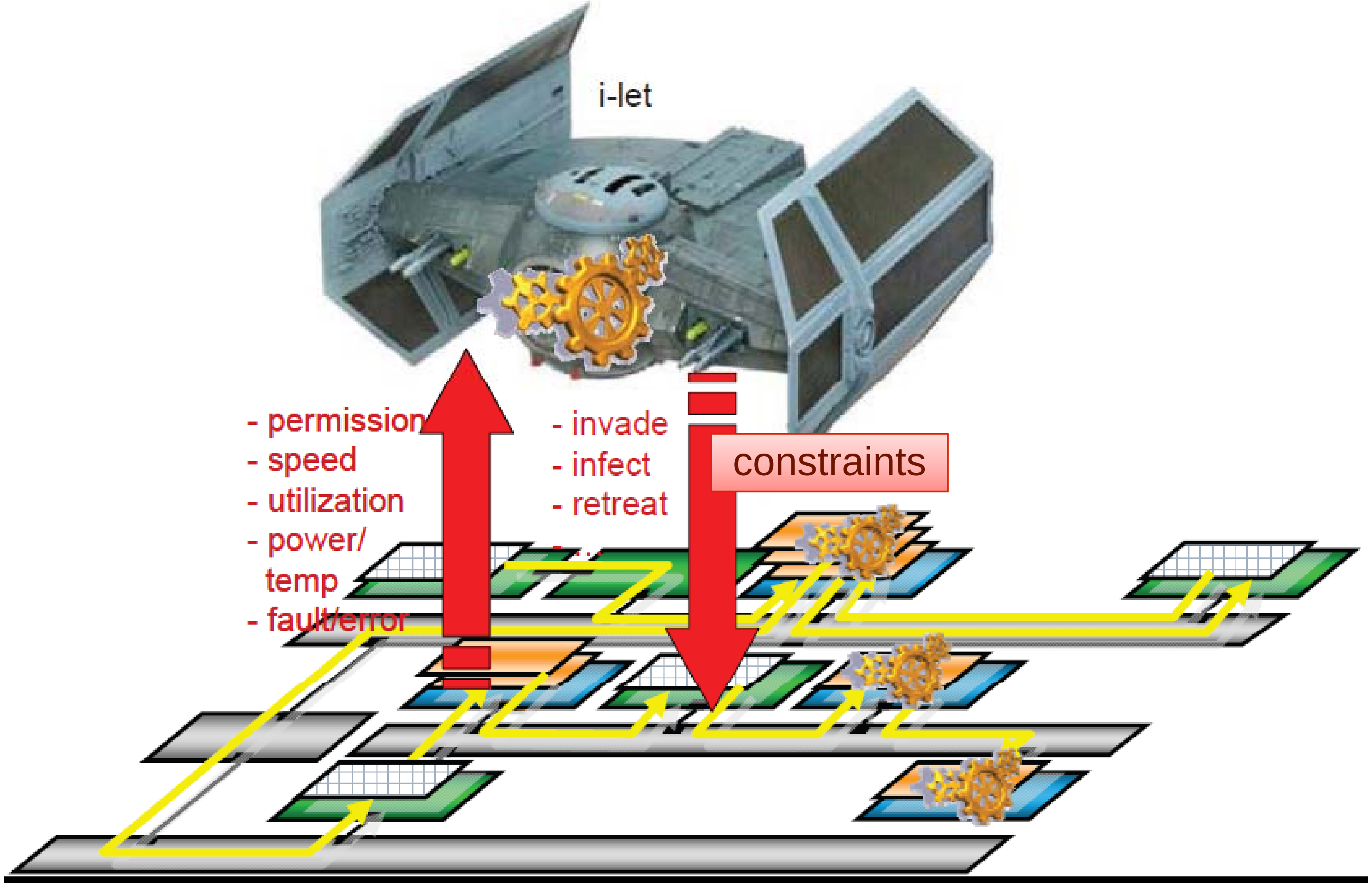
Source: [2]

Hello World in X10

```
public class HelloWorld {
  public static def main(args:Array[String](1)) {
    finish for(p in Place.places())
      async at (p)
        Console.OUT.println(“Hello from place ”+here.id);
  }
}
```

```
$ x10c++ -o hello HelloWorld.x10
$ mpirun -n 4 hello
Hello from place 0
Hello from place 2
Hello from place 3
Hello from place 1
```

Invasive Programming Concepts



Invasive Programming Library

- **Basic control flow**

```
val claim = Claim.invade(constraints);
claim.infect(ilet);
claim.retreat();
```

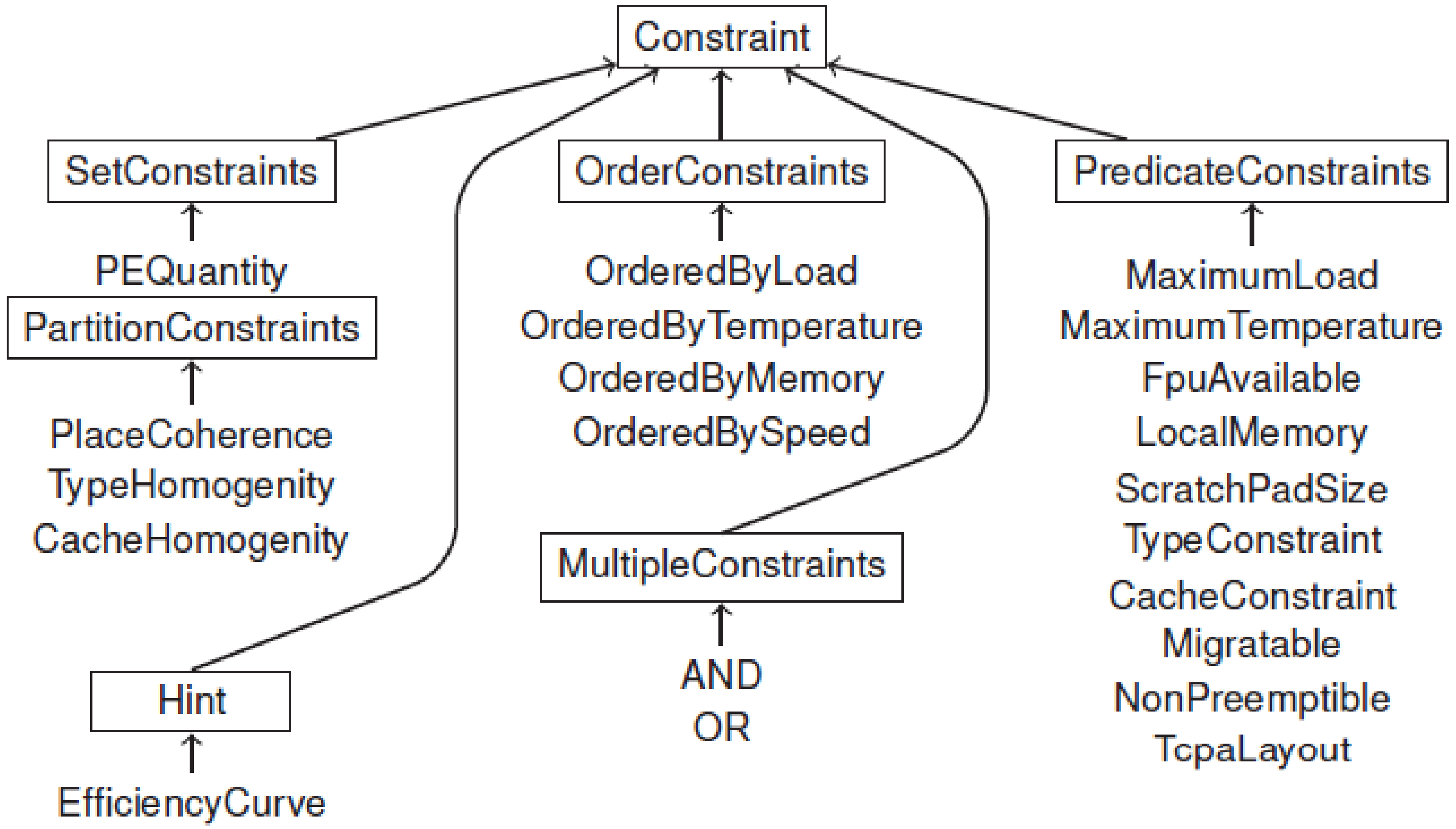
- **Constraints**

```
val constraints = new AND();
constraints.add(new PEQuantity(1,8));
constraints.add(new PlaceCoherence());
constraints.add(new MaximumLoad(0.7f));
```

- **i-lets**

```
val ilet = (id:IncarnationID) => {
  Console.OUT.println(“Hello from ilet “+id);
};
```

Constraint Hierarchy



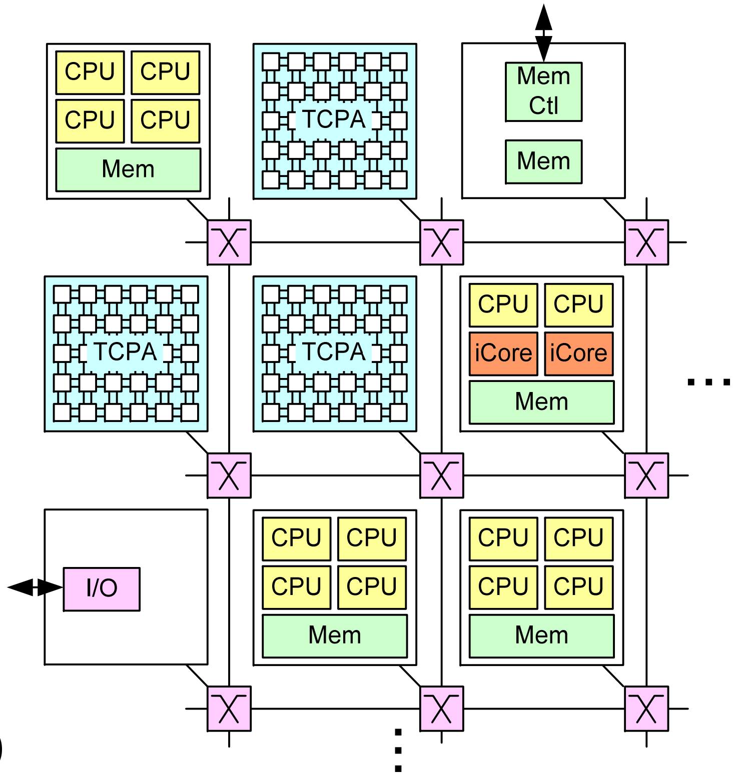
Example: Color Space Transformation

```
val img = Image.load(filename);
val constraint = new AND();
constraint.add(new TypeConstraint(PEType.TCPA));
constraint.add(new PEQuantity(1));
constraint.add(new TCPALayout(10,10));

try {
  val claim = Claim.invade(constraints);
  // parallel execution
  claim.infect((id:IncarnationID) => {
    ComponentTransform.forwardIctTCPA(img);
  });
} catch (e:NotEnoughResources) {
  // local execution
  ComponentTransform.forwardIctCPU(img);
}
```

Project Area B – Architectures

- Invasive Computer Architectures:
  - Invasion Control Architectures for networks of ASIP- (iCore [B1]), RISC- (CPU [B3]) and Tightly-Coupled 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]

Outline

- Invasive Programming in X10
  - Introduction to X10
  - Invasive Programming Library

- Simulation of Invasive Programs and MPSoC architectures
  - Goals
  - Simulation Model
  - Case Study
  - Future Work

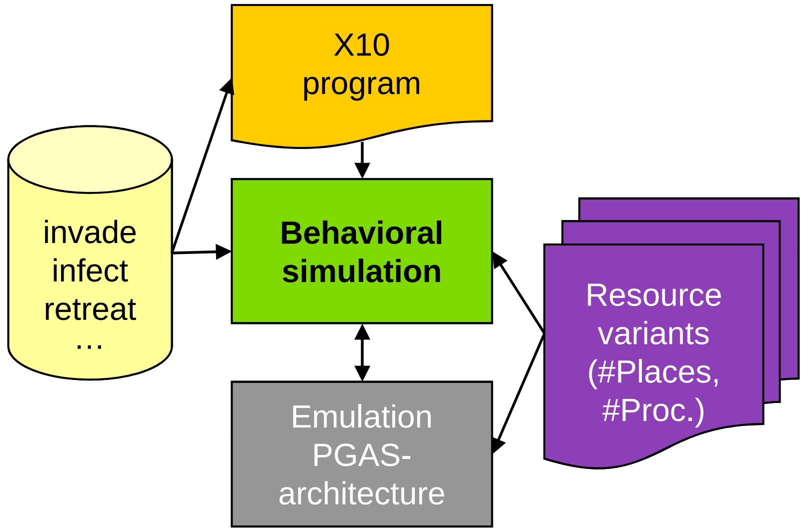
Functional Simulation

- **Goals:**
  - Enables early validation of invasive programming concepts
  - Allows the investigation of a broad range of different hardware platforms
  - Full hardware and software implementations are not yet available

- **Purpose:**
  - Application programmer
    - Learn to think invasively
    - Analyze benefits of resource-aware programming
  - Architectural designers
    - Explore different invasive architectures
  - Operating systems engineers
    - Investigate invasion strategies

Design and Implementation

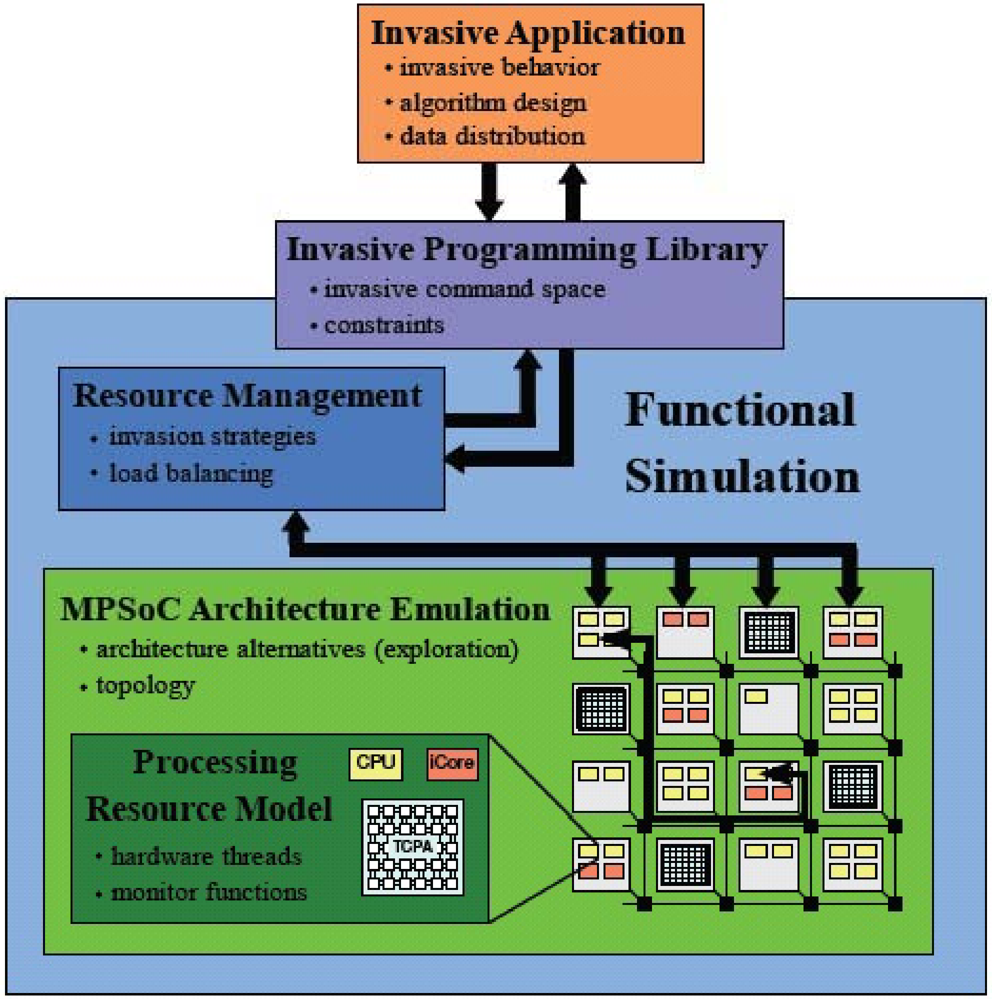
- **Design Decisions:**
  - Functional level, not cycle-accurate
    - Otherwise much too slow
  - Realization of the main commands
    - invade
    - infect
    - Retreat
  - Rudimentary architecture emulation
  - Fully X10-based
    - Highly parallel and distributed implementation
    - Light-weight threads


X10 program
invade
infect
retreat
…
Behavioral simulation
Resource variants (#Places, #Proc.)
Emulation PGAS-architecture

- **Current Restrictions:**
  - Only processing resources model
    - No communication, or I/O resources model yet
  - No timing model yet

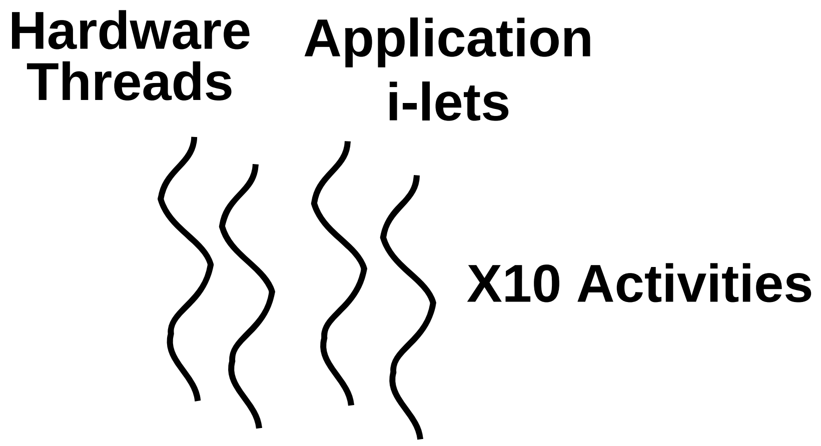
„Big Picture“ of the Functional Simulator

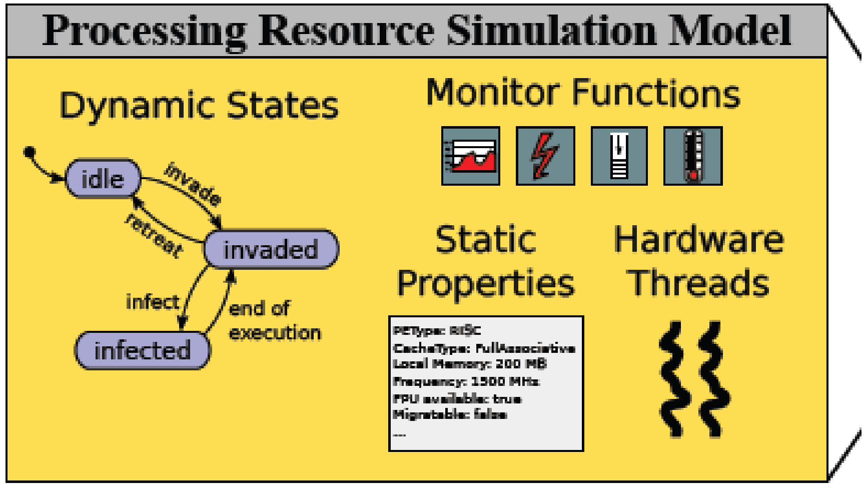
- **Components:**
  - Application level
  - Invasive programming library
  - Resource management
  - MPSoC architecture emulation
    - Emulation through a concept called “Hardware Threads”



Processing Resource Simulation Model

- **Hardware Threads:**
  - Encapsulate all important HW state information
  - Interact with the runtime system
  - Realized by X10 activities
- **Static Properties:**
  - PEType
  - Local Memory
- **Dynamic States:**
  - Functionality realized by a state machine
  - Events cause state changes
- **Monitor Functions:**
  - Simulate physical or logical states of a processing resource
  - Temperature, Load, Power Consumption, Faultiness

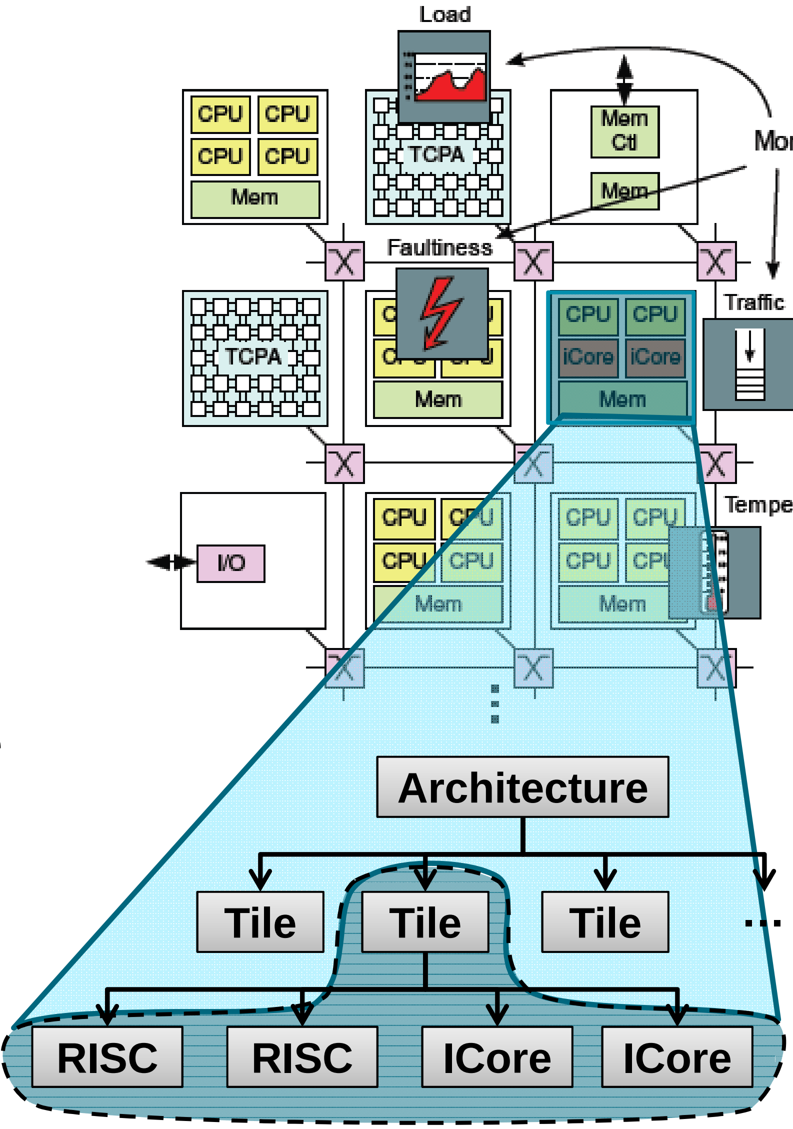




PEType: RISC
CacheType: FullAssociative
Local Memory: 200 MB
Frequency: 1500 MHz
FPU available: true
Migratable: false
...

Simulation of Invasion on MPSoCs

- **Properties:**
  - Tiled architecture
  - Connected via a NoC
  - Heterogeneous compute tiles
    - RISC
    - iCore
    - TCPA
  - Augmented with monitor information
  - Topology
- **MPSoC Modeling:**
  - Using the previous processing resource simulation model
  - Mapping on a class hierarchy
  - Architecture is the root
  - Each tile consists of several processing elements
  - Emulation through hardware threads



MPSoC Modeling Example

```
// create a new architecture
val arch = new Architecture();
// create a new tile within this architecture
val tile = arch.createTile();
// create four RISC CPUs within the tile
for (i:Int=0; i<4; i++) {
  val pe = tile.createRISC();
  // specify the properties of the RISC CPU
  pe.peType = PEType.RISC;
  pe.cacheType = CacheType.FourWayAssociative;
  pe.localMem = 2048; // KiB
  pe.scratchPadMem = 128; // KiB
  pe.clockFrequency = 1500; // MHz
  pe.isMigratable = false;
  pe.isPreemptible = false;
  pe.hasFPU = true;
}
```

Simulation

- **Simulator**
  - Global configuration of the simulation environment
    - Topology size, invasion strategy, …
  - Initializes and activates the emulated architecture

```
Simulator.init(args);
```

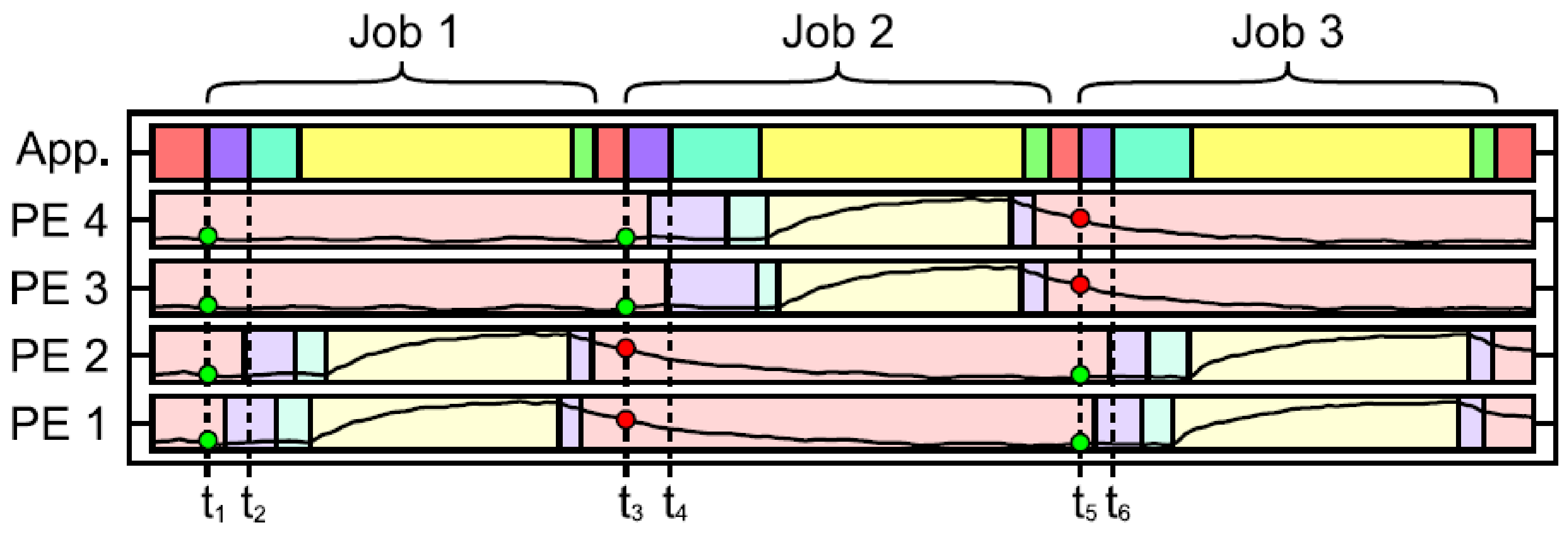
  - Starts applications
    - With a certain `delay` of ms
    - At a particular tile `address` within the topology

```
Simulator.startApplication(app, 500, new GridAddress(1,2));
```

  - Exits the emulated architecture and shuts down the created activities

```
Simulator.exit();
```

Case Study: Temperature-Aware Load Balancing



- Four PE tile
- Three job batch processing application
  - Allocating two PEs
- Maximum Temperature Constraint: 70°C

States of the application:

idle	🟥
invading	🟪
infecting	🟦
executing	🟨
retreating	🟩

States of the PEs:

idle	🟥
invaded	🟪
infected	🟦
running	🟨

Current and Future Work

- **Current Work:**
  - Invasive computing idea provides resource-aware programming facilities
  - Library-based language implementation of invasive computing using the X10 programming language
    - Extension of X10 instead of designing a new language
  - Framework to compile and simulate resource-aware programs on emulated MPSoC platforms
    - Early simulation facilities in order to validate the invasive programming constructs

- **Future Work:**
  - Extension of the framework for the modeling of the allocation of
    - Memories
    - Communication resources
  - Provide a proper timing model
    - Design space exploration of invasive applications and architectures becomes possible

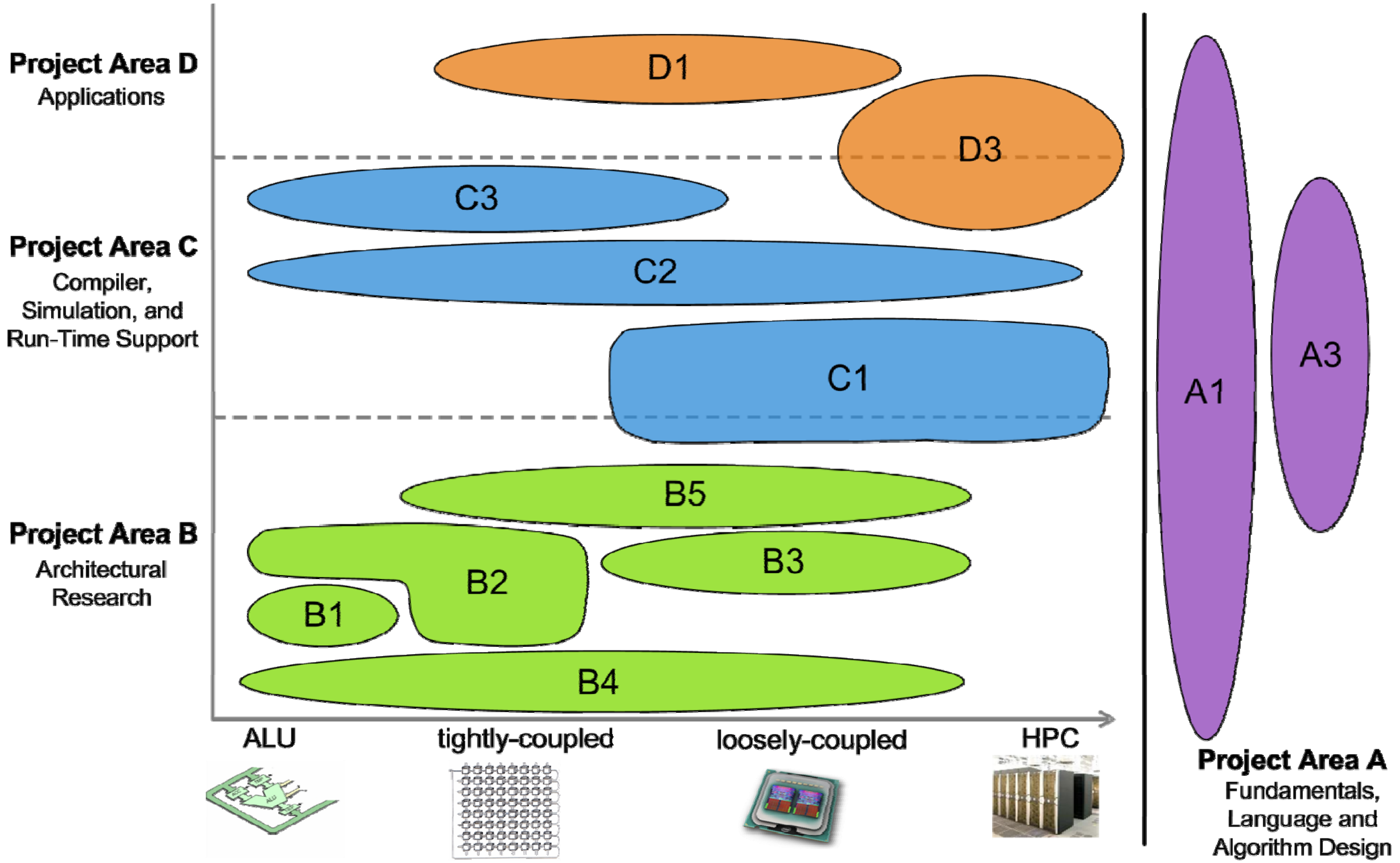
Project Area D – Applications

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

Outline

- 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: Robotics, Scientific Computing
- Structure, Chances and Goals
  - Project structure
  - Funded Institutions and Researchers
  - Demonstrator Roadmap
  - Impact and Risks

TRR 89 – Project Structure

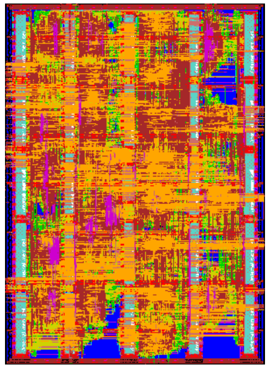


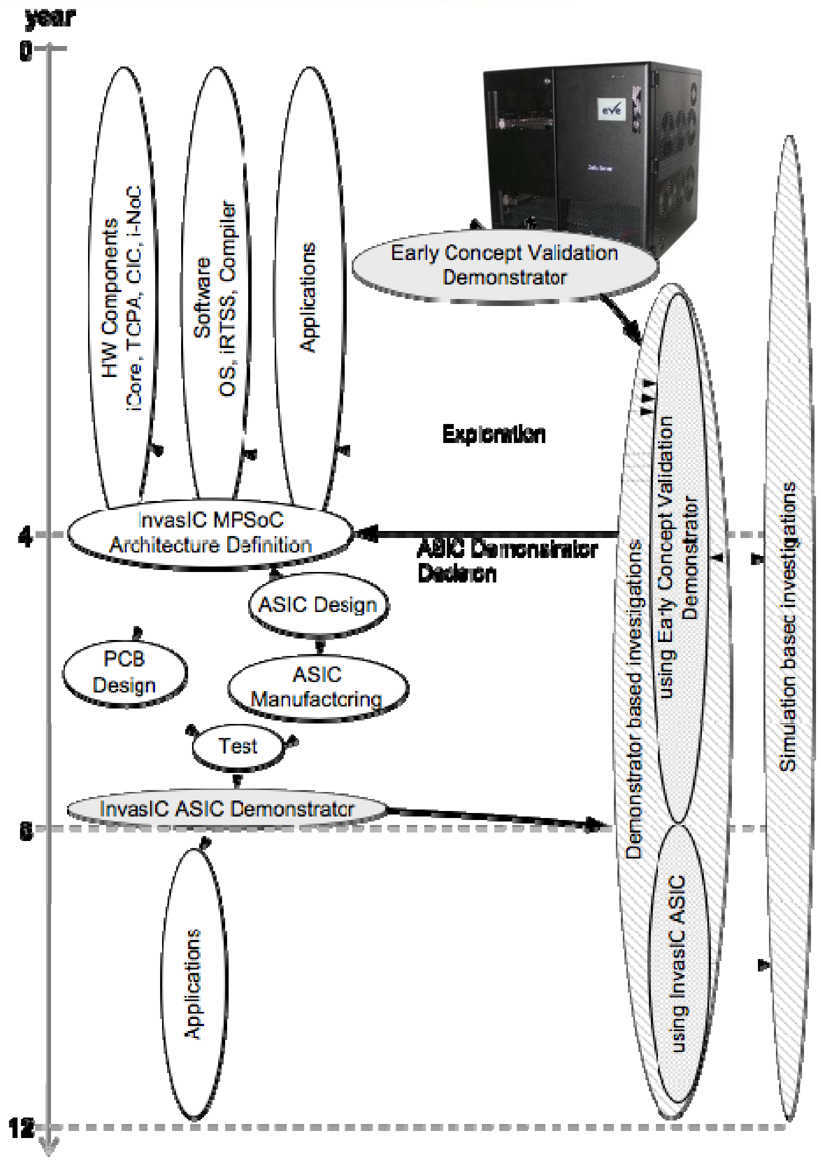
TRR 89 – Funded Institutions and Researchers

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**
A3: Scheduling and Load Balancing **Sanders**	**B2: Invasive Tightly-Coupled Processor Arrays** **Teich**	**C2: Simulation of Invasive Applications and Invasive Architectures** **Hannig/Gerndt/Herkersdorf**	**D3: Multilevel Approaches and Adaptivity in Scientific Computing** **Bungartz/Gerndt**
	B3: Invasive Loosely-Coupled MPSoC **Herkersdorf/Henkel**	**C3: Compilation and Code Generation for Invasive Programs** **Snelting/Teich**	
	B4: Hardware Monitoring System and Design Optimization for Invasive Architectures **Schmitt-Landsiedel/Schlichtmann**		
	B5: Invasive NoCs **Becker/Herkersdorf/Teich**		

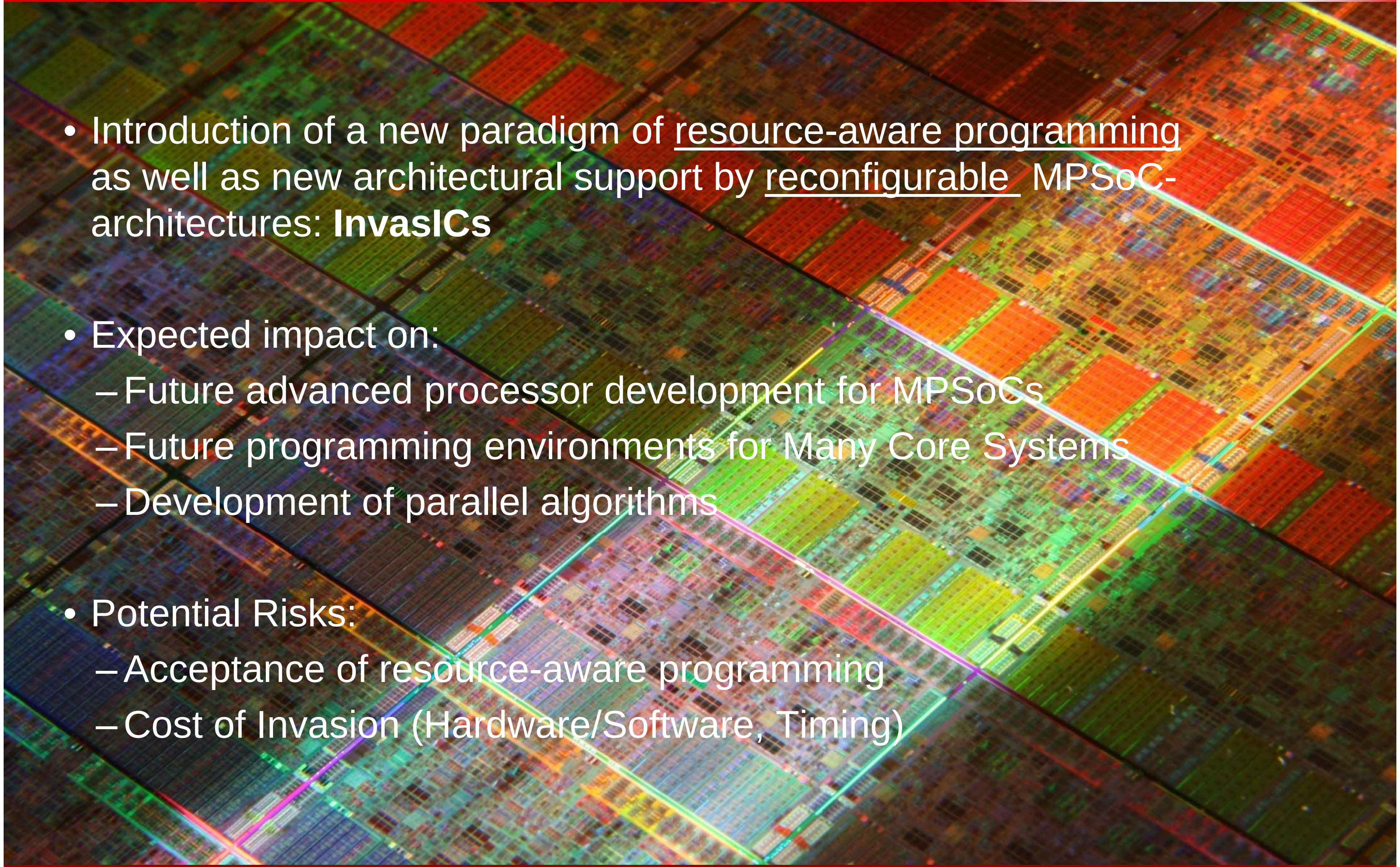
TRR 89 – Validation & Demonstrator Roadmap

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




year
0
4
8
12
HW Components
iCore, TCPA, CIC, i-NoC
Software
OS, iRTSS, Compiler
Applications
Early Concept Validation Demonstrator
Exploration
InvasIC MPSoC Architecture Definition
ASIC Demonstrator Decision
ASIC Design
PCB Design
ASIC Manufactcring
Test
InvasIC ASIC Demonstratcr
Applications
Demonstrator based investigations
using Early Concept Validation Demonstrator
using InvasIC ASIC
Simulation based investigations

Impact and Risks DFG TRR 89 InvasIC



- 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)