Java Environment for Parallel Realtime Development

Echtzeit-Java für Mehrkernsysteme

Java Forum Stuttgart
3. Juli 2008
Dr. Fridtjof Siebert, CTO, aicas GmbH, Karlsruhe

Outline

- Project Overview
- Work Structure
- Correctness of Parallel Applications

Jeopard Project Overview

- ECs 7th Framework Programme
- Timeframe:
  Jan 2008 – June 2010
- Management lead:
  The Open Group
- Technological lead:
  aicas GmbH
- Total Budget: 3.3Mio€

Jeopard Project Overview

- 10 Project Partners

aicas realtime

SYSGO EMBEDDING INNOVATIONS

THE Open GROUP Making standards work®

THE UNIVERSITY of York

UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA

FZI

RadioLabs

TU WIEN

EADS

SKYSOFT PORTUGAL

Project Goal

- Provide a platform independent software development environment for
  - complex,
  - safe,
  - realtime,
  - multicore systems.
- Leverage off existing technology and past projects:
  - Java, RTSJ (JSR 1&282), SC-Java (JSR 302)
  - AJACS, HIDOORS, HIJA, FRESCOR

Multilayered Approach

- Applications
- Tools
- API (Java and C)
- Java VM
- Operating Systems (RTOS)
- CPU Architecture

Project Work Structure

Project Work Structure

WP 2: Architecture Layer

Project Work Structure

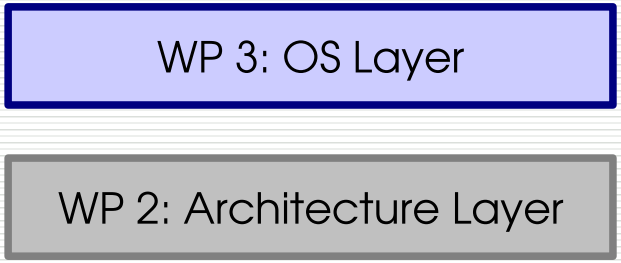
WP 2: Architecture Layer

Project Work Structure

- Multi-Core Java Processor (JOP)
- Synchronization between Processors
- Future Non-Uniform-Memory-Architectures

WP 2: Architecture Layer

Project Work Structure



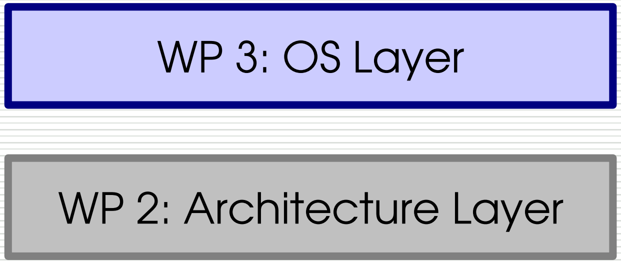
WP 3: OS Layer

WP 2: Architecture Layer

Project Work Structure

UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA

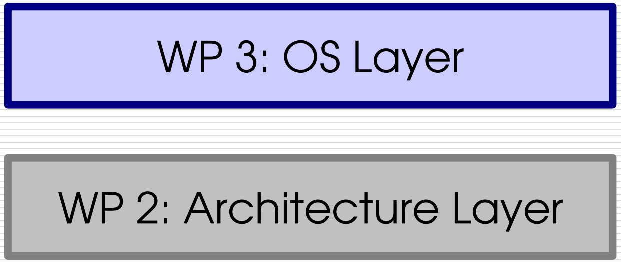
SYSGO EMBEDDING INNOVATIONS


WP 3: OS Layer

WP 2: Architecture Layer

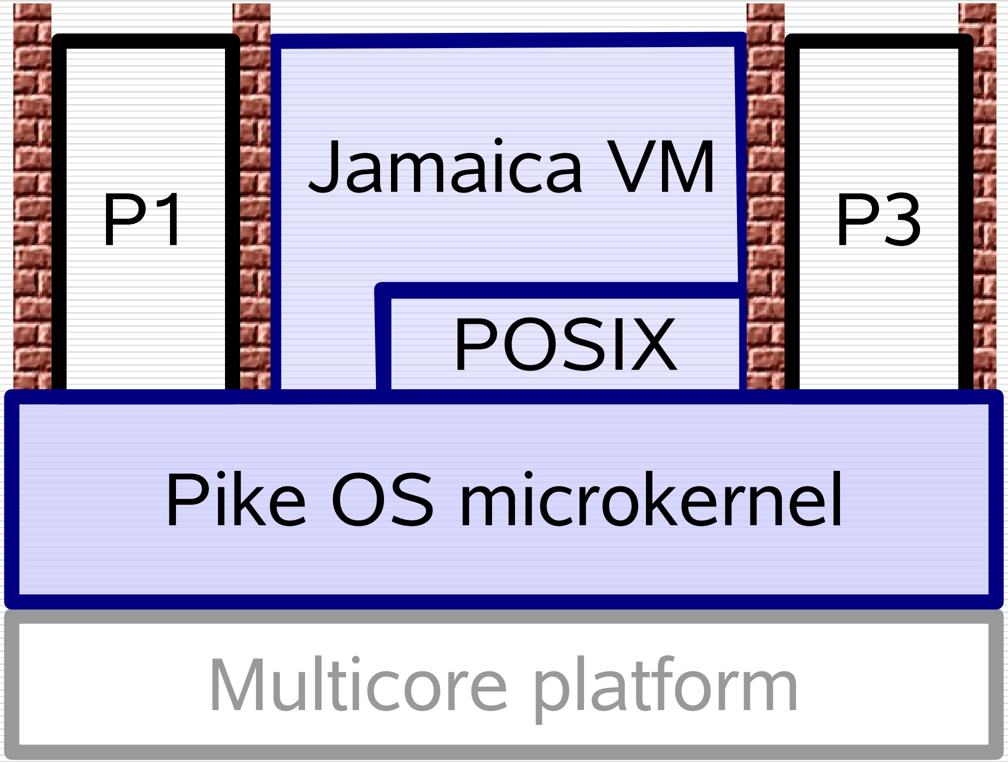
Project Work Structure

- Parallel Partitioning RTOS
- HW Abstraction Layer
- Mapping: Java Threads ↔ OS Threads



Project Work Structure

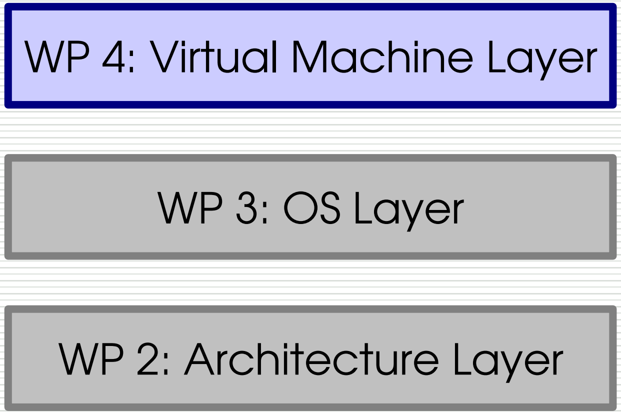
- Example for Partitioning



UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA

SYSGO EMBEDDING INNOVATIONS

Project Work Structure

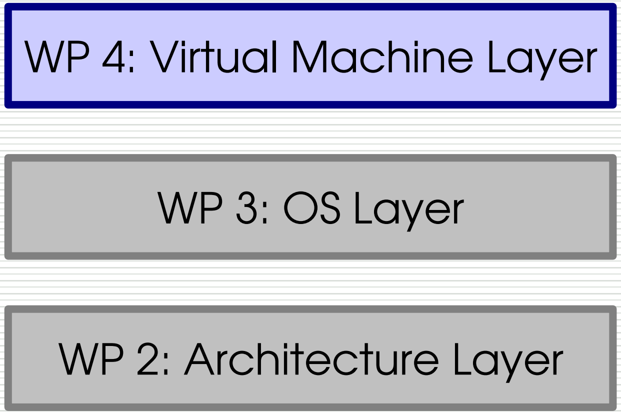


WP 4: Virtual Machine Layer

WP 3: OS Layer

WP 2: Architecture Layer

Project Work Structure



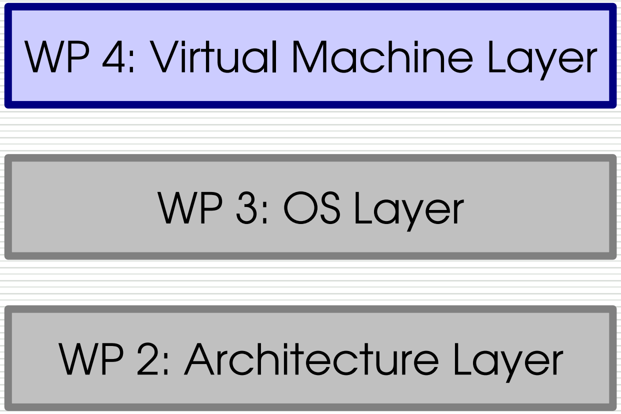
WP 4: Virtual Machine Layer

WP 3: OS Layer

WP 2: Architecture Layer

Project Work Structure

- Parallel Realtime JVM
- Parallel Realtime GC
- Parallel Monitors etc.

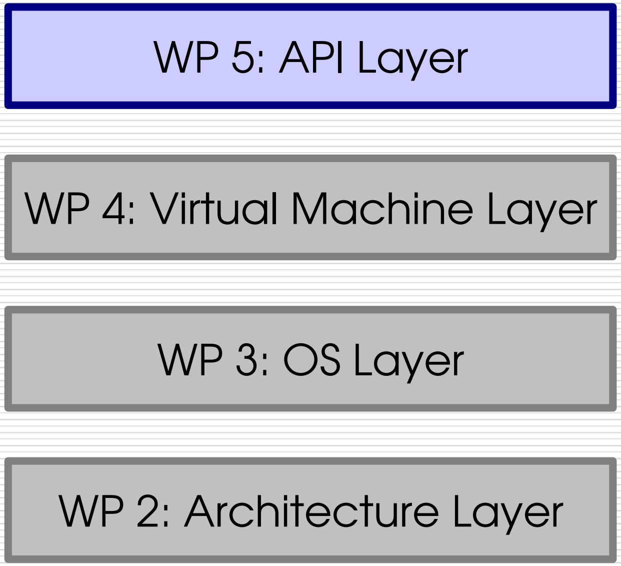


WP 4: Virtual Machine Layer

WP 3: OS Layer

WP 2: Architecture Layer

Project Work Structure



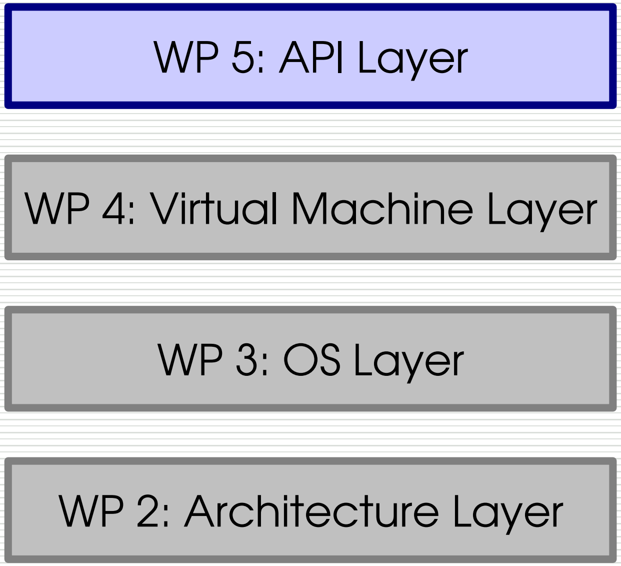
WP 5: API Layer

WP 4: Virtual Machine Layer

WP 3: OS Layer

WP 2: Architecture Layer

Project Work Structure



WP 5: API Layer

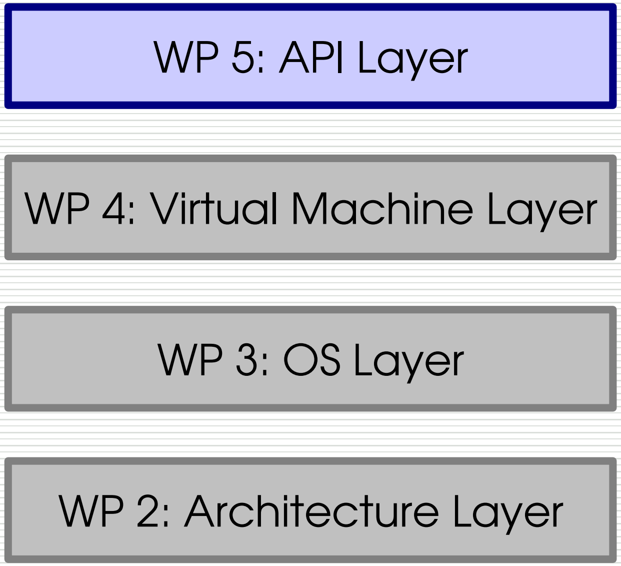
WP 4: Virtual Machine Layer

WP 3: OS Layer

WP 2: Architecture Layer

Project Work Structure

- Multicore OS-level APIs
- Multicore Java-API extensions (RTSJ)
- Standardization

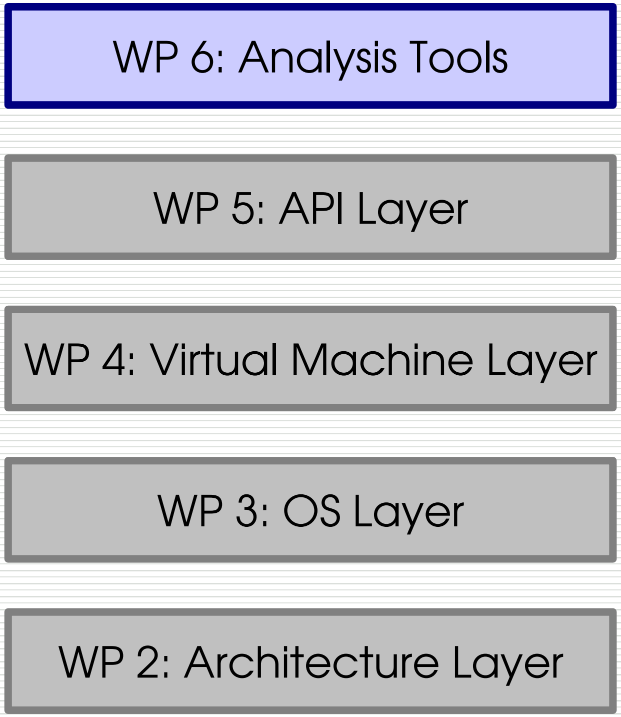

WP 5: API Layer

WP 4: Virtual Machine Layer

WP 3: OS Layer

WP 2: Architecture Layer

Project Work Structure



WP 6: Analysis Tools

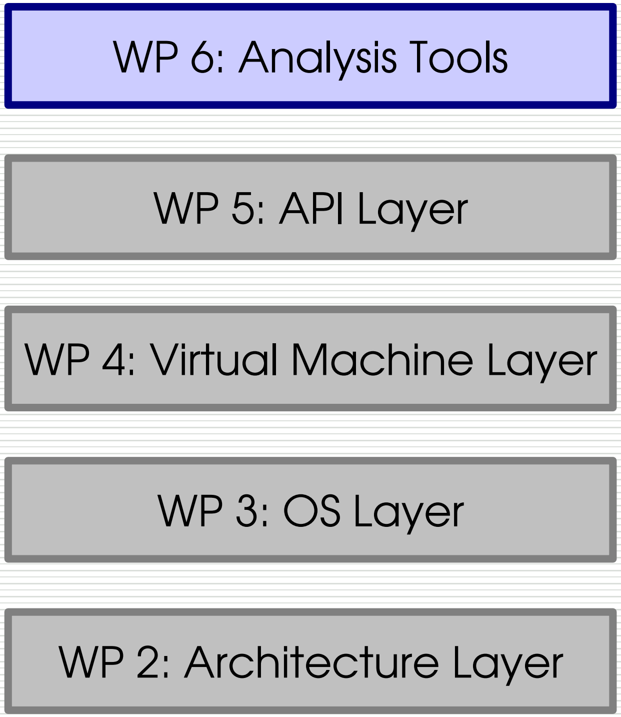
WP 5: API Layer

WP 4: Virtual Machine Layer

WP 3: OS Layer

WP 2: Architecture Layer

Project Work Structure



WP 6: Analysis Tools

WP 5: API Layer

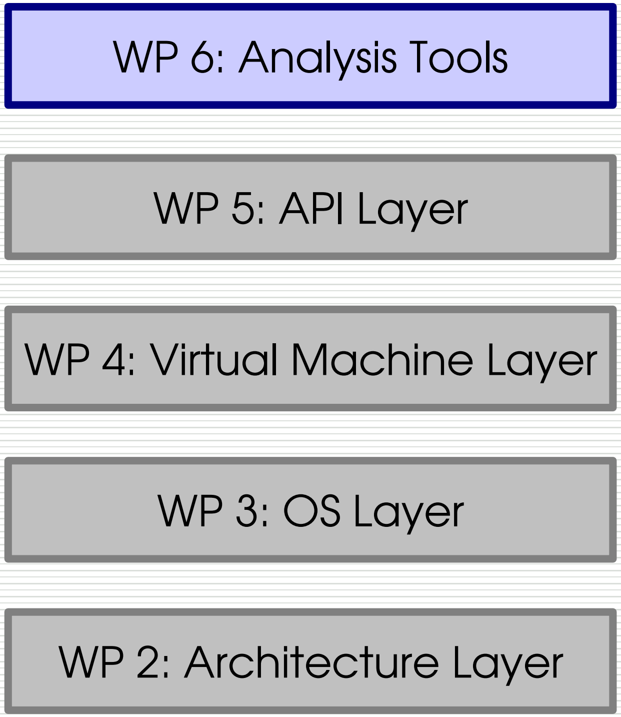
WP 4: Virtual Machine Layer

WP 3: OS Layer

WP 2: Architecture Layer

Project Work Structure

- Static Analysis
- Fresco Contract Model for Java
- Concurrent Unit Testing



WP 6: Analysis Tools

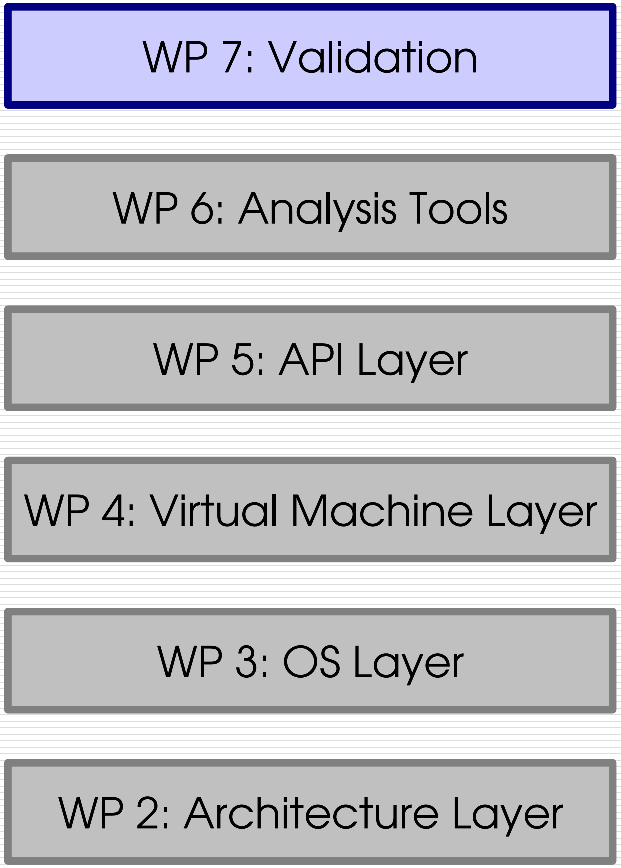
WP 5: API Layer

WP 4: Virtual Machine Layer

WP 3: OS Layer

WP 2: Architecture Layer

Project Work Structure



Project Work Structure

WP 7: Validation

WP 6: Analysis Tools

WP 5: API Layer

WP 4: Virtual Machine Layer

WP 3: OS Layer

WP 2: Architecture Layer

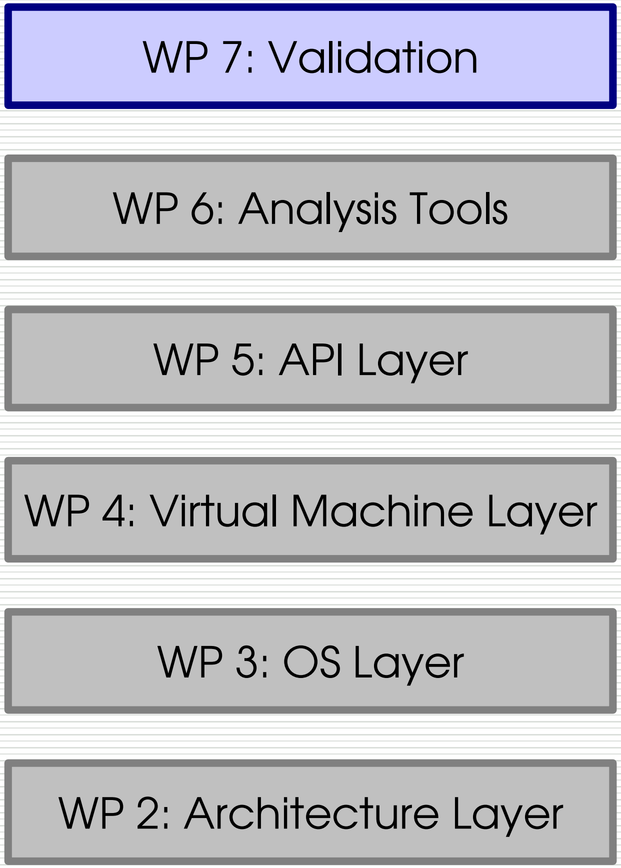
RadioLabs

EADS

SKYSOFT PORTUGAL

Project Work Structure

- Multicore Radar
- SW Radio
- Onboard Aircraft Control


WP 7: Validation

WP 6: Analysis Tools

WP 5: API Layer

WP 4: Virtual Machine Layer

WP 3: OS Layer

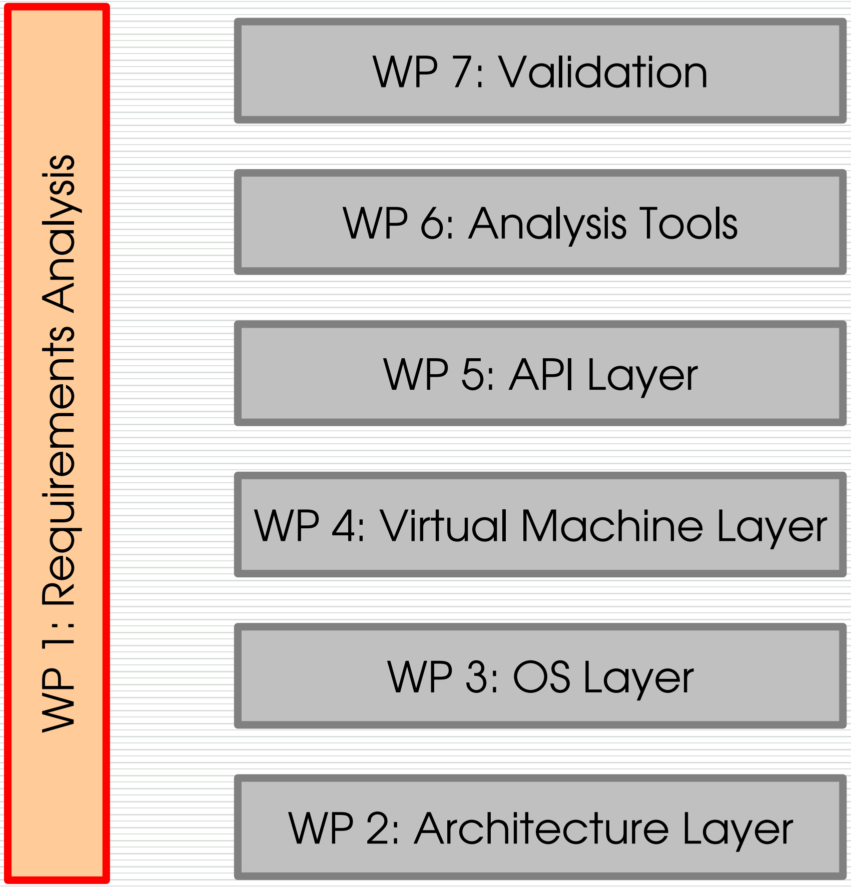
WP 2: Architecture Layer

RadioLabs

EADS

SKYSOFT PORTUGAL

Project Work Structure



WP 1: Requirements Analysis

WP 7: Validation

WP 6: Analysis Tools

WP 5: API Layer

WP 4: Virtual Machine Layer

WP 3: OS Layer

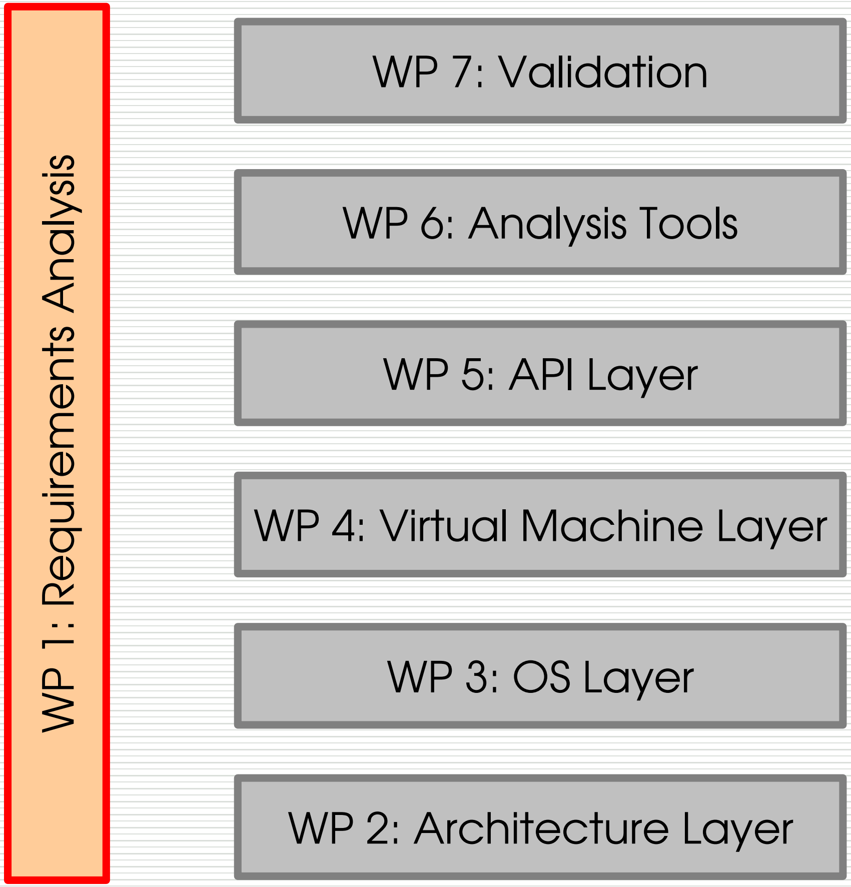
WP 2: Architecture Layer

Project Work Structure

EADS

RadioLabs

SKYSOFT
PORTUGAL



WP 1: Requirements Analysis

WP 7: Validation

WP 6: Analysis Tools

WP 5: API Layer

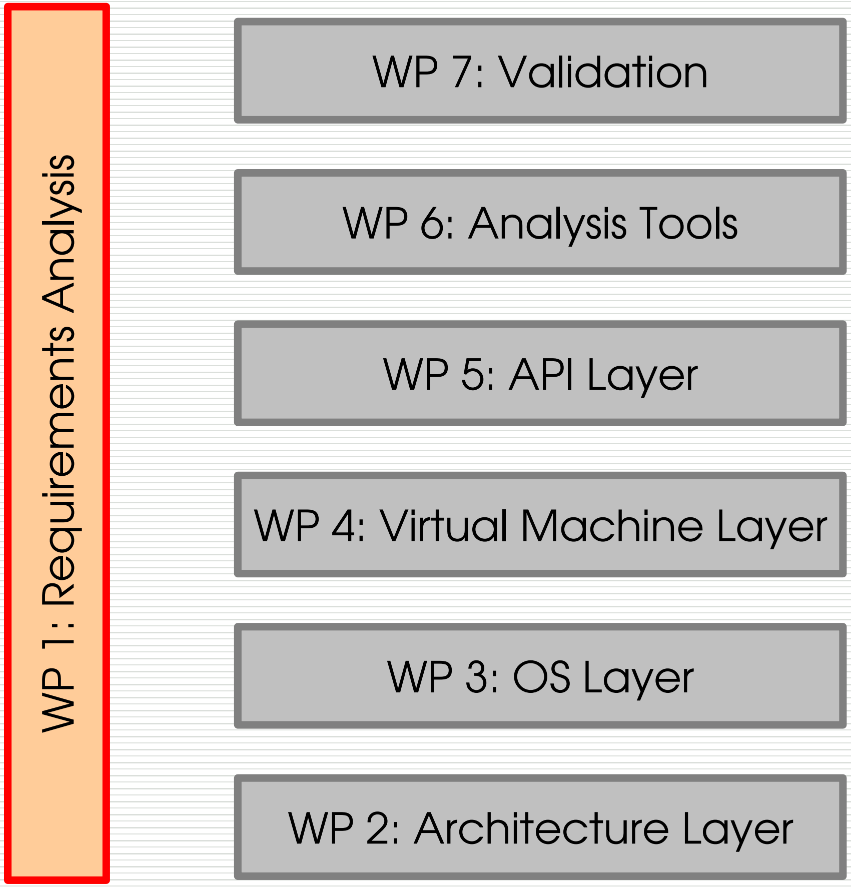
WP 4: Virtual Machine Layer

WP 3: OS Layer

WP 2: Architecture Layer

Project Work Structure

- Application driven:



WP 1: Requirements Analysis

WP 7: Validation

WP 6: Analysis Tools

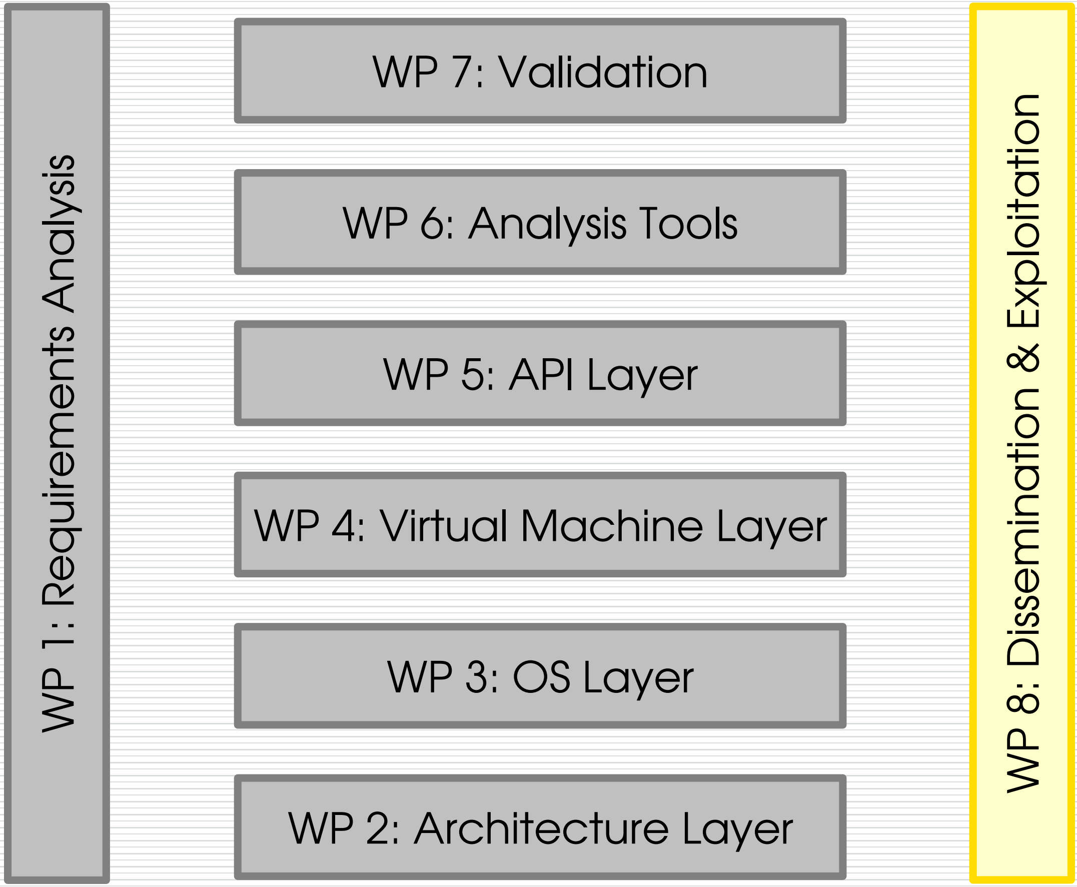
WP 5: API Layer

WP 4: Virtual Machine Layer

WP 3: OS Layer

WP 2: Architecture Layer

Project Work Structure



WP 1: Requirements Analysis

WP 7: Validation

WP 6: Analysis Tools

WP 5: API Layer

WP 4: Virtual Machine Layer

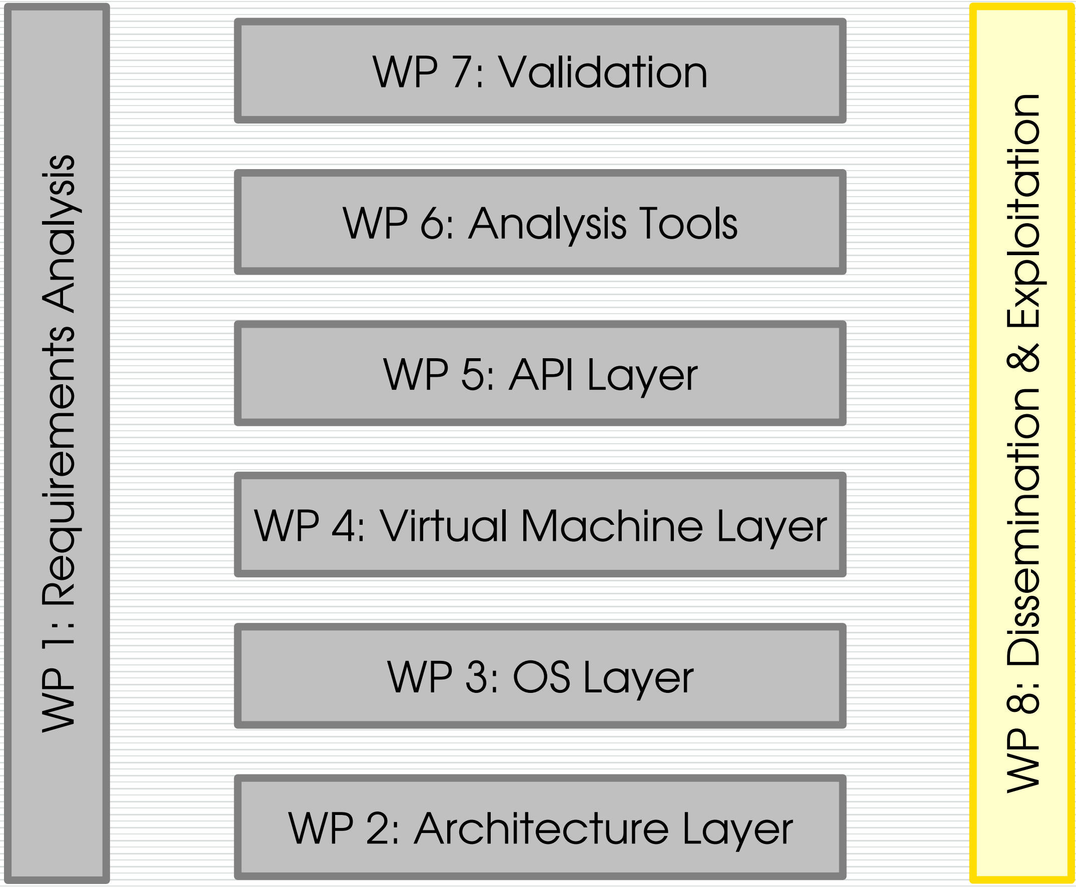
WP 3: OS Layer

WP 2: Architecture Layer

WP 8: Dissemination & Exploitation

Project Work Structure

- Results Public
  - standards
  - open source
  - products



WP 1: Requirements Analysis

WP 7: Validation

WP 6: Analysis Tools

WP 5: API Layer

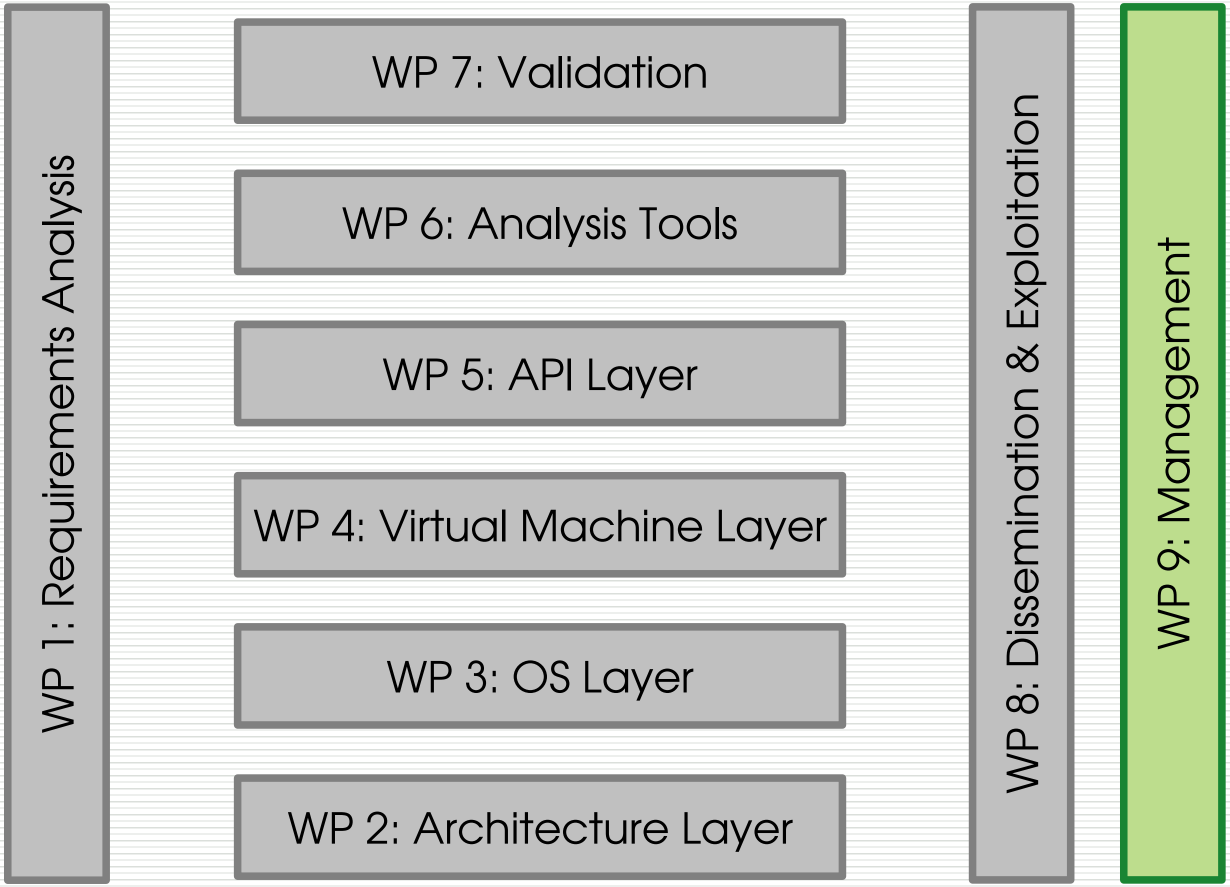
WP 4: Virtual Machine Layer

WP 3: OS Layer

WP 2: Architecture Layer

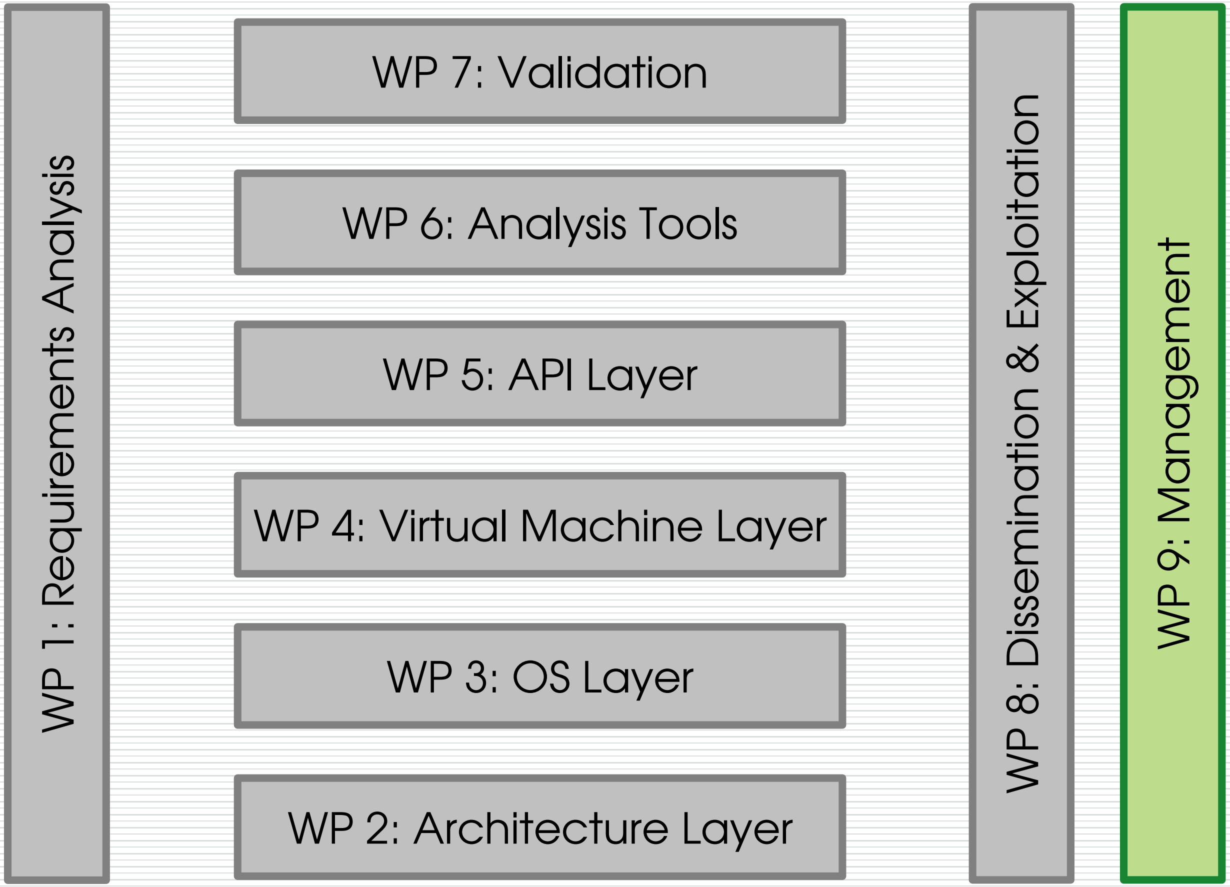
WP 8: Dissemination & Exploitation

Project Work Structure



Project Work Structure

- Strict steering



WP 1: Requirements Analysis

WP 7: Validation

WP 6: Analysis Tools

WP 5: API Layer

WP 4: Virtual Machine Layer

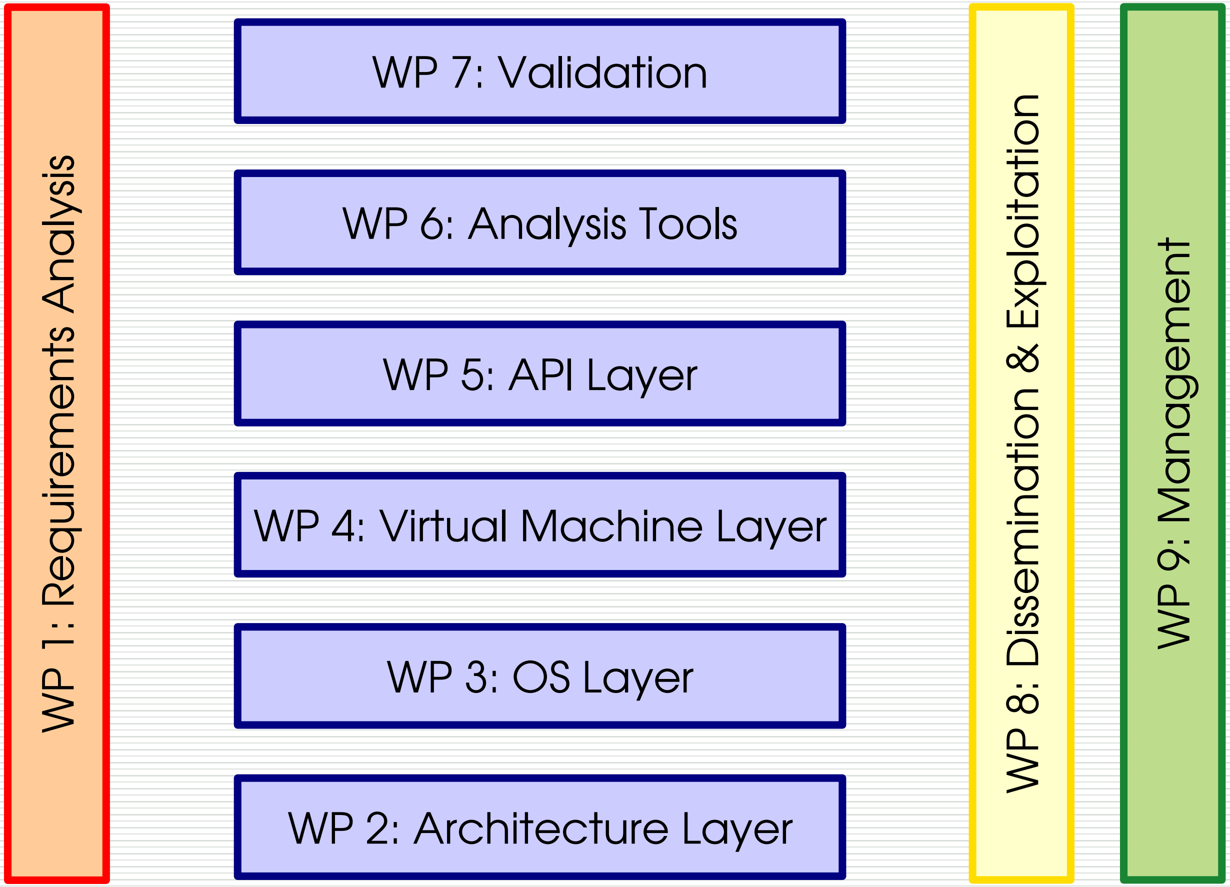
WP 3: OS Layer

WP 2: Architecture Layer

WP 8: Dissemination & Exploitation

WP 9: Management

Project Work Structure



WP 1: Requirements Analysis

WP 7: Validation

WP 6: Analysis Tools

WP 5: API Layer

WP 4: Virtual Machine Layer

WP 3: OS Layer

WP 2: Architecture Layer

WP 8: Dissemination & Exploitation

WP 9: Management

What does JEOPARD bring to you?

New Java APIs

- RTSJ Extensions for multi-core systems
  - CPU affinity:

    ```
    realtimeThread.setAffinity(bitset)
    ```

    - restrict thread to given CPU(s)
    - be unaffected by threads on different CPUs
    - avoid slowdown due to simultaneous multithreading

New Java APIs

- Additional Parallel APIs
  - Parallel *forAll*:

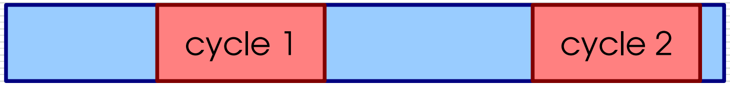
```
new ParallelSet(set).forAll(actn);
```

- permit parallel execution
- leave the details (# CPUs, # threads, etc.) open
- make actual assignment of threads, CPUs, priorities on target

Realtime Garbage Collection

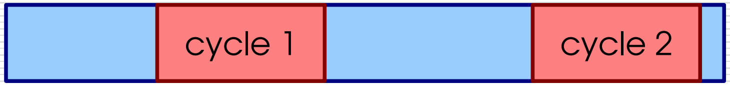
Realtime Garbage Collection

Blocking GC

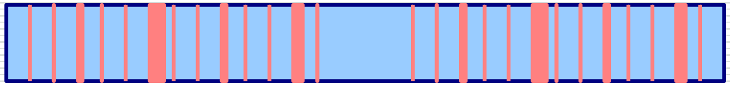


Realtime Garbage Collection

Blocking GC

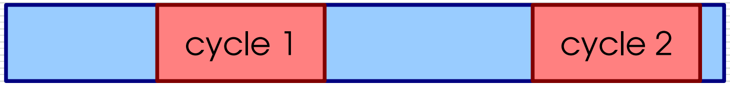


Incremental GC



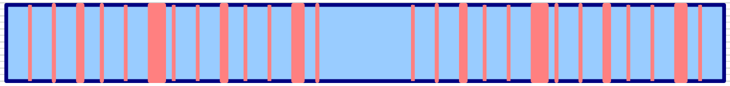
Realtime Garbage Collection

Blocking GC

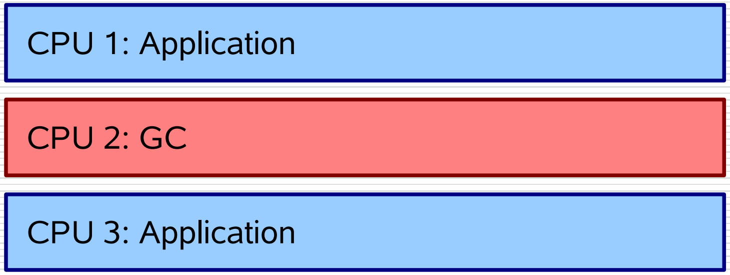

cycle 1

cycle 2

Incremental GC



Concurrent GC

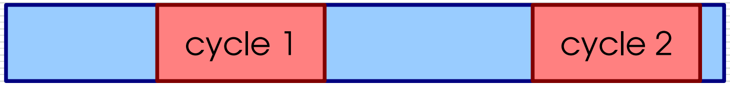

CPU 1: Application

CPU 2: GC

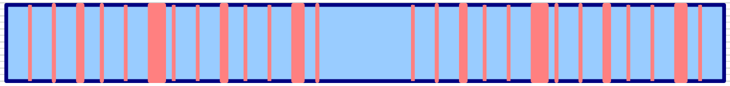
CPU 3: Application

Realtime Garbage Collection

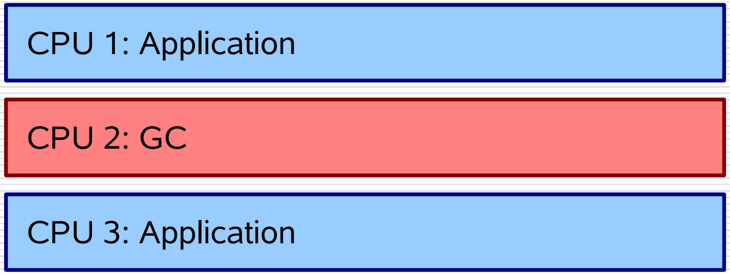
Blocking GC


cycle 1
cycle 2

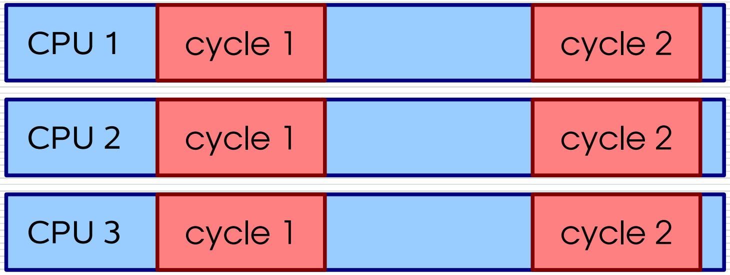
Incremental GC



Concurrent GC

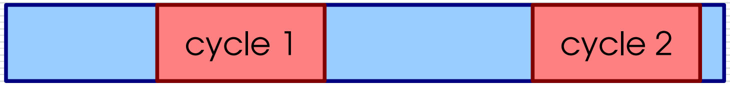

CPU 1: Application
CPU 2: GC
CPU 3: Application

Parallel GC


CPU 1 cycle 1 cycle 2
CPU 2 cycle 1 cycle 2
CPU 3 cycle 1 cycle 2

Realtime Garbage Collection

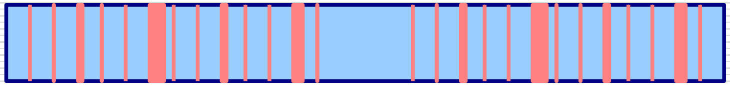
Blocking GC



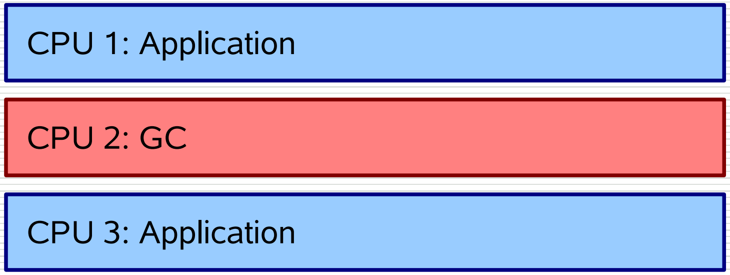
cycle 1

cycle 2

Incremental GC



Concurrent GC

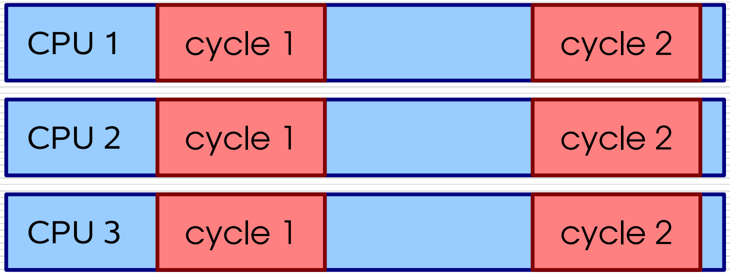


CPU 1: Application

CPU 2: GC

CPU 3: Application

Parallel GC

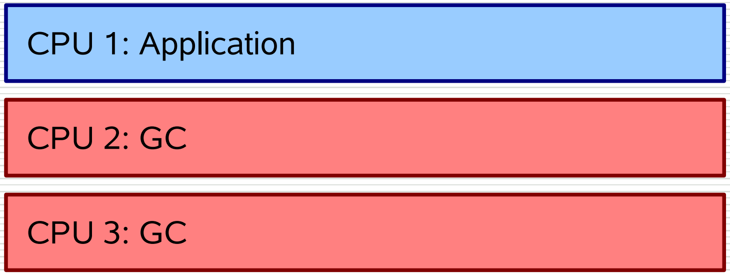


CPU 1 | cycle 1 | cycle 2

CPU 2 | cycle 1 | cycle 2

CPU 3 | cycle 1 | cycle 2

Parallel & Concurrent GC



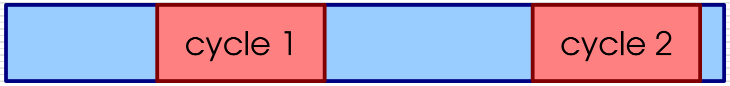
CPU 1: Application

CPU 2: GC

CPU 3: GC

Realtime Garbage Collection

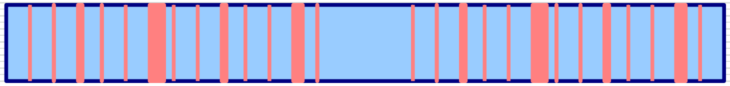
Blocking GC



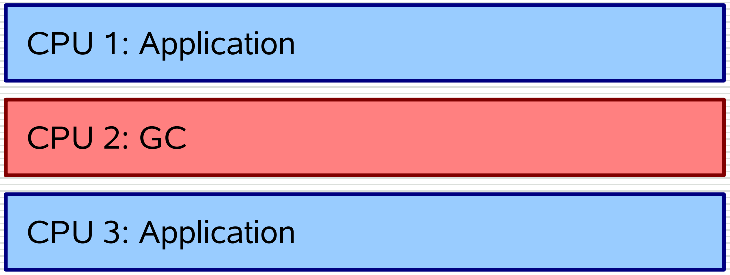
cycle 1

cycle 2

Incremental GC



Concurrent GC

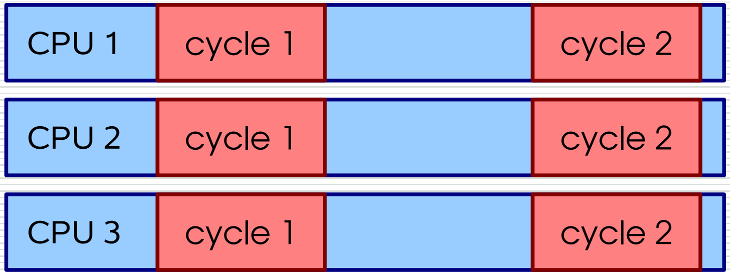


CPU 1: Application

CPU 2: GC

CPU 3: Application

Parallel GC

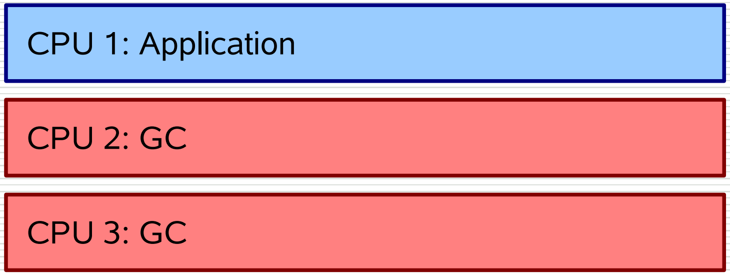


CPU 1 cycle 1 cycle 2

CPU 2 cycle 1 cycle 2

CPU 3 cycle 1 cycle 2

Parallel & Concurrent GC

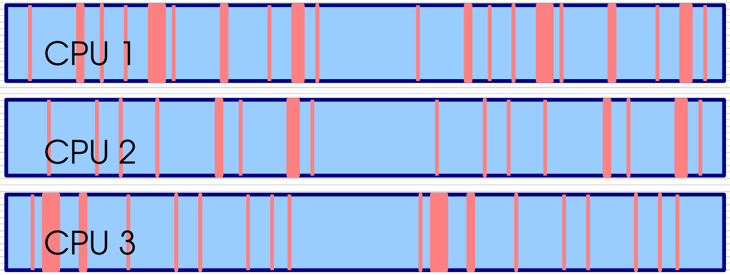


CPU 1: Application

CPU 2: GC

CPU 3: GC

Parallel Realtime GC



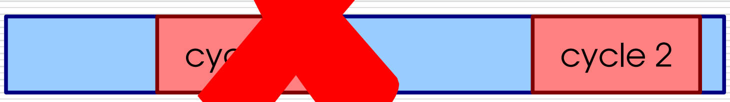
CPU 1

CPU 2

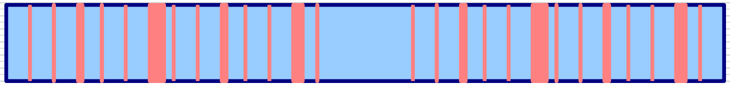
CPU 3

Realtime Garbage Collection

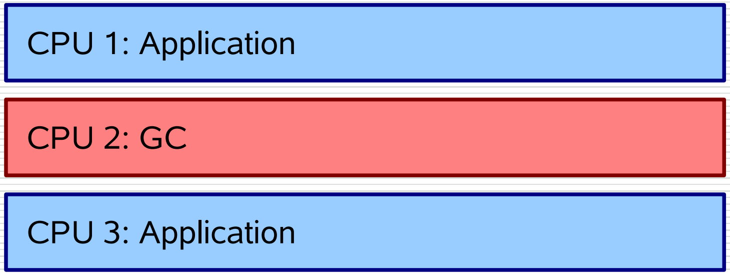
~~Blocking GC~~



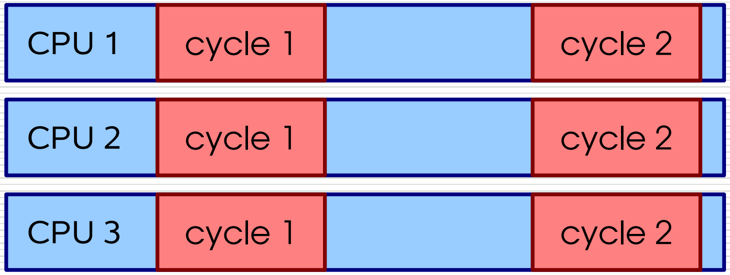
Incremental GC



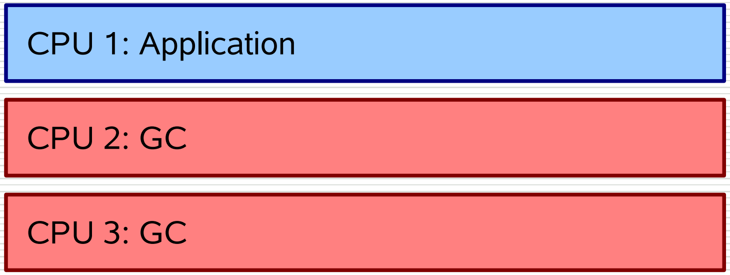
Concurrent GC



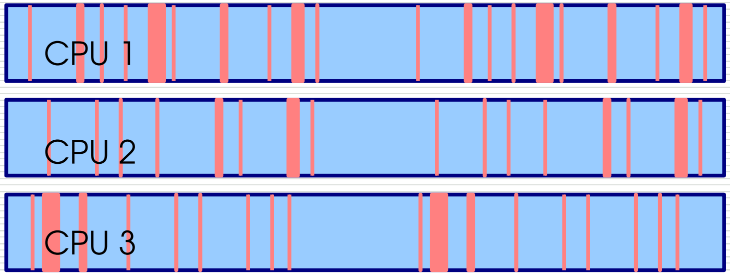
Parallel GC



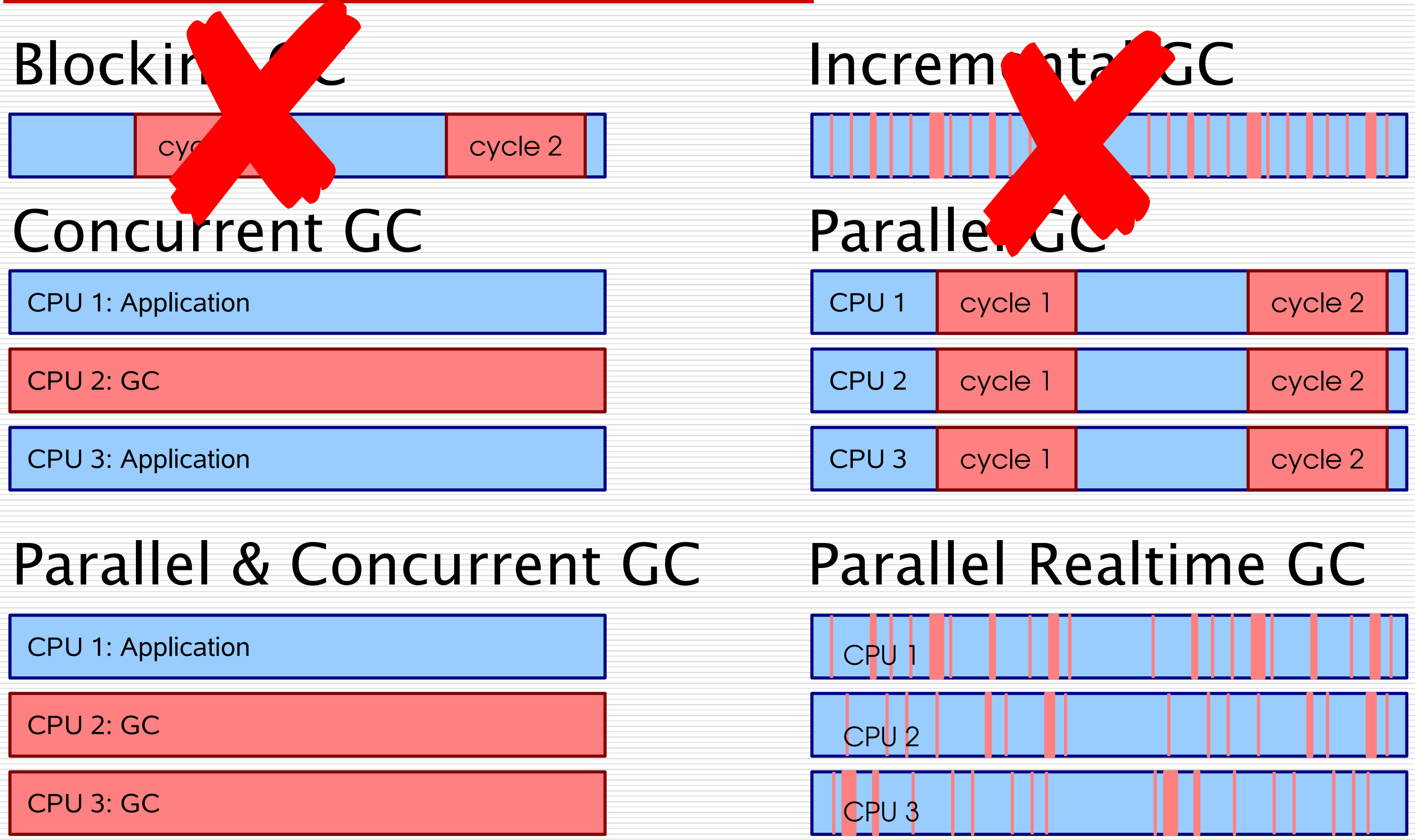
Parallel & Concurrent GC



Parallel Realtime GC



Realtime Garbage Collection



Blocking GC

cycle 1

cycle 2

Incremental GC

Concurrent GC

CPU 1: Application

CPU 2: GC

CPU 3: Application

Parallel GC

CPU 1

cycle 1

cycle 2

CPU 2

cycle 1

cycle 2

CPU 3

cycle 1

cycle 2

Parallel & Concurrent GC

CPU 1: Application

CPU 2: GC

CPU 3: GC

Parallel Realtime GC

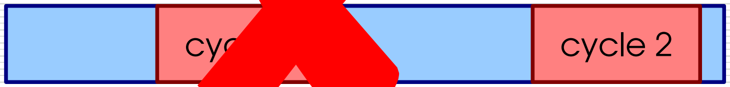
CPU 1

CPU 2

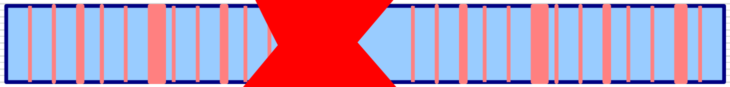
CPU 3

Realtime Garbage Collection

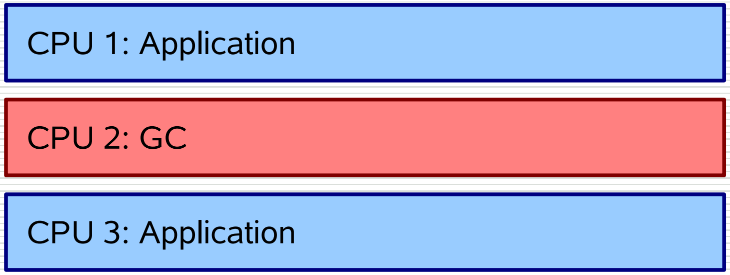
Blocking GC



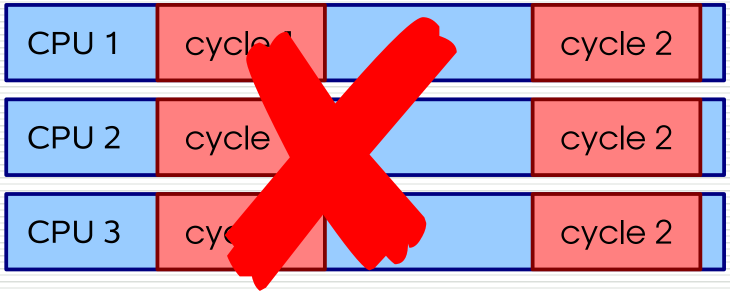
Incremental GC



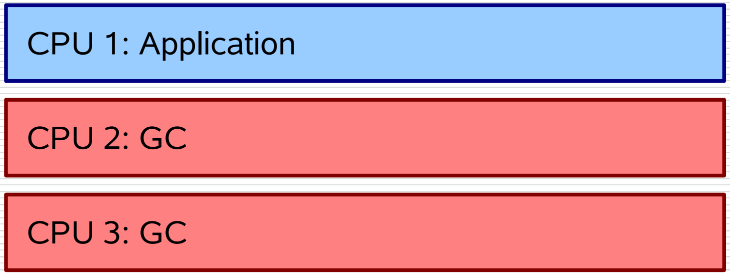
Concurrent GC



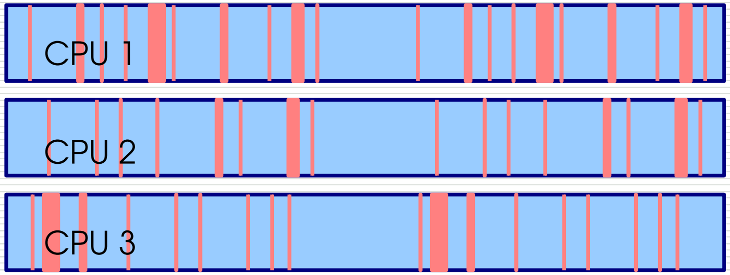
Parallel GC



Parallel & Concurrent GC

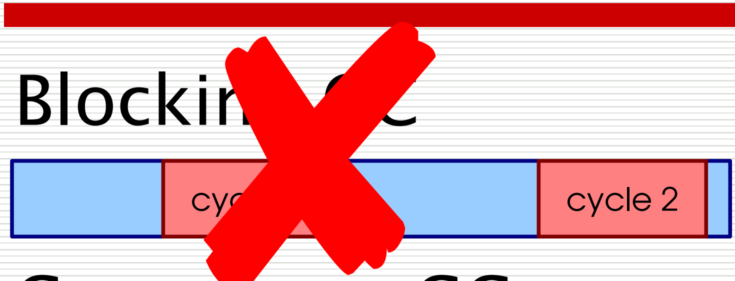


Parallel Realtime GC

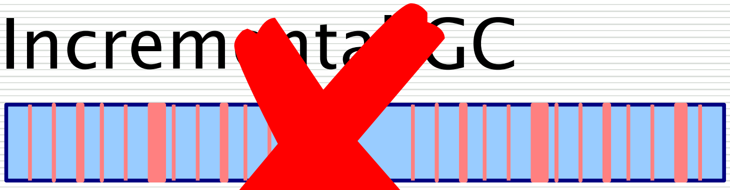


Realtime Garbage Collection

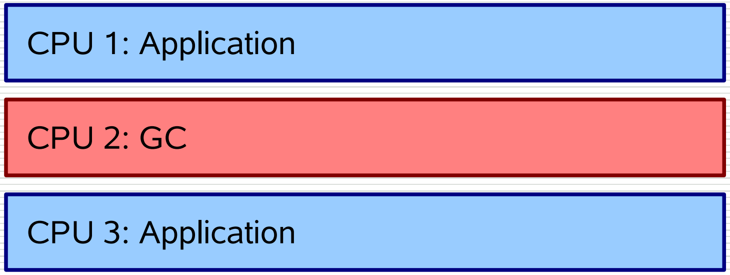
Blocking GC



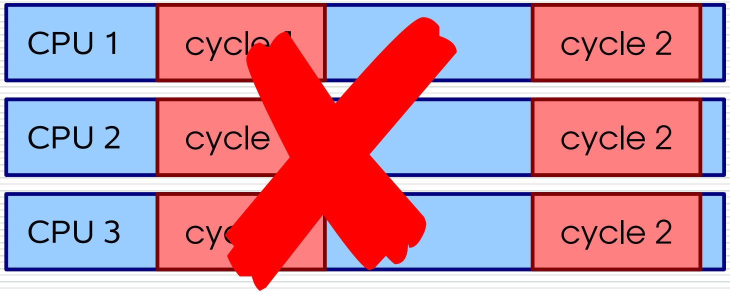
Incremental GC



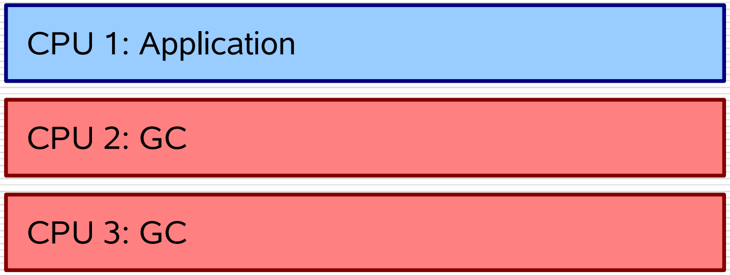
Concurrent GC



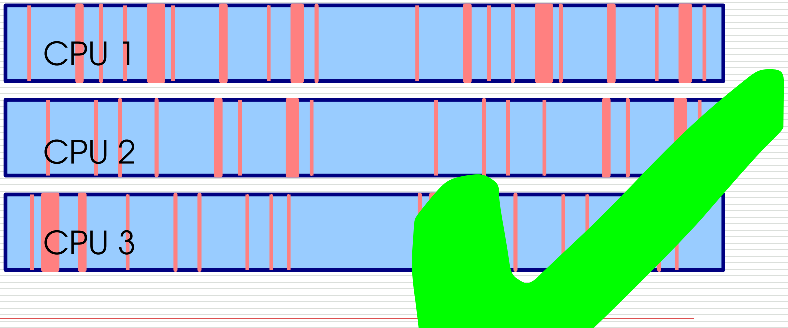
Parallel GC



Parallel & Concurrent GC



Parallel Realtime GC



Realtime Garbage Collection

- First Results
  - Full parallel GC not always possible
  - Need shallow heap graph!
  - GC implementation has to avoid contention:
    - use CPU local structures
    - uses compare-and-set on different memory locations
  - Free list representation under investigation

Typical Programming Errors

- Classical Race Conditions
  - Thread 1 / Thread 2

Thread 1	Thread 2
`obj.i++;`	`obj.i++;`

Typical Programming Errors

- Classical Race Conditions
  - Thread 1 — Thread 2

```
obj.i++;                obj.i++;
```

  - variable *i* might be incremented only once!
  - Failure is very unlikely on single-CPU!
  - Likelihood on multi-core is much higher
  - 'Heisenbug': If you try to look at it, it disappears

Typical Programming Errors

- Classical Race Conditions
  - Thread 1 / Thread 2

```
hashmap.put(a,b);      x = hashmap.get(y);
```

Typical Programming Errors

- Reordering

```
static Whatever ref = null;
static boolean refSet = false;
```

Typical Programming Errors

- Reordering

```
static Whatever ref = null;
static boolean refSet = false;
```

  - Thread 1:

```
ref = new Whatever();
```

Typical Programming Errors

- Reordering

```
static Whatever ref = null;
static boolean refSet = false;
```

  - Thread 1:

```
ref = new Whatever();
refSet = true;
```

Typical Programming Errors

- Reordering

```
static Whatever ref = null;
static boolean refSet = false;
```

  - Thread 1:

```
ref = new Whatever();
refSet = true;
```

  - Thread 2:

```
if (refSet) ref.doSomething();
```

Typical Programming Errors

- Reordering

```
static Whatever ref = null;
static boolean refSet = false;
```

  - Thread 1:

```
ref = new Whatever();
refSet = true;
```

  - Thread 2:

```
if (refSet) ref.doSomething();
```

  - possible NullPointerException!

Typical Programming Errors

- Reordering

```
static Whatever ref = null;
static boolean refSet = false;
```

  - Thread 1: legal exeuction sequence:

```
ref = new Whatever();   tmp = new Whatever();
refSet = true;          refSet = true;
                        ref = tmp;
```

  - Thread 2:

```
if (refSet) ref.doSomething();
```

  - possible NullPointerException!

JEOPARD Solutions

- Static Data-Flow Analysis
  - flag all detected race conditions

JEOPARD Solutions

- Static Data-Flow Analysis
  - flag all detected race conditions
  - but it will not detect logical errors:

Thread 1:

```
if (hashtable.containsKey(x))
  {

    hashtable.get(x).doXYZ();

  }
```

JEOPARD Solutions

- Static Data-Flow Analysis
  - flag all detected race conditions
  - but it will not detect logical errors:

Thread 1:

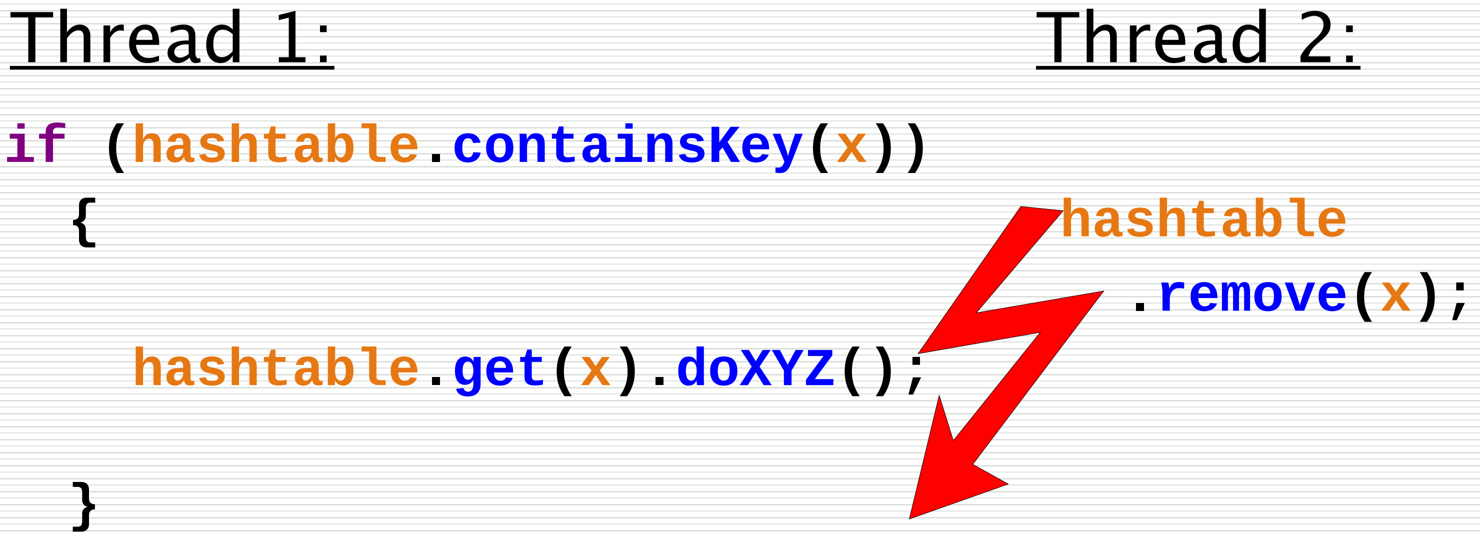
```
if (hashtable.containsKey(x))
  {
    hashtable.get(x).doXYZ();

  }
```

Thread 2:

```
hashtable
  .remove(x);
```

JEOPARD Solutions

- Static Data-Flow Analysis
  - flag all detected race conditions
  - but it will not detect logical errors:



Thread 1:

```
if (hashtable.containsKey(x))
  {

    hashtable.get(x).doXYZ();

  }
```

Thread 2:

```
hashtable
  .remove(x);
```

JEOPARD Solutions

- Parallel Unit Testing
  - test possible execeution paths for JUnit tests
  - automatically generate coverage of relevant parallel interleavings

Conclusion

- JEOPARD Currently at the end of the Requirements Phase
- Design and Development is starting
- We expect first project-internal prototypes late in 2008
- Full toolchain and validation results in mid 2010

Conclusion

- Results will become available at

  www.jeopard.org

- or contact me:

  siebert@aicas.com