

# A class-based programming model for heterogeneous MPSoC

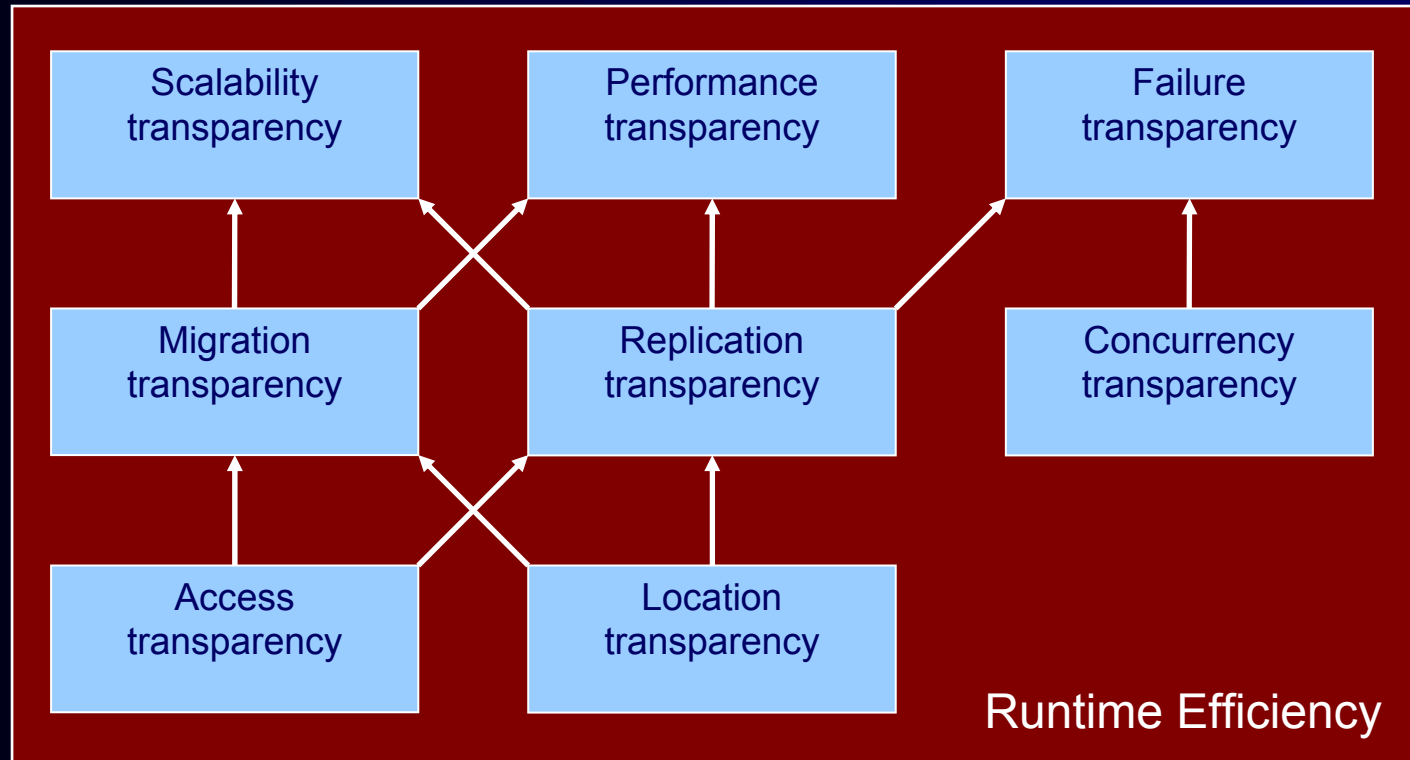
MPSoC '05

Mark Lippett, Ignios Ltd.

# Agenda

- Existing trends and techniques
- The Unified Kernel Layer
- Example: RPC over UKL
- Performance
- Conclusions

# Judging the “goodness” of an MPSoC programming model



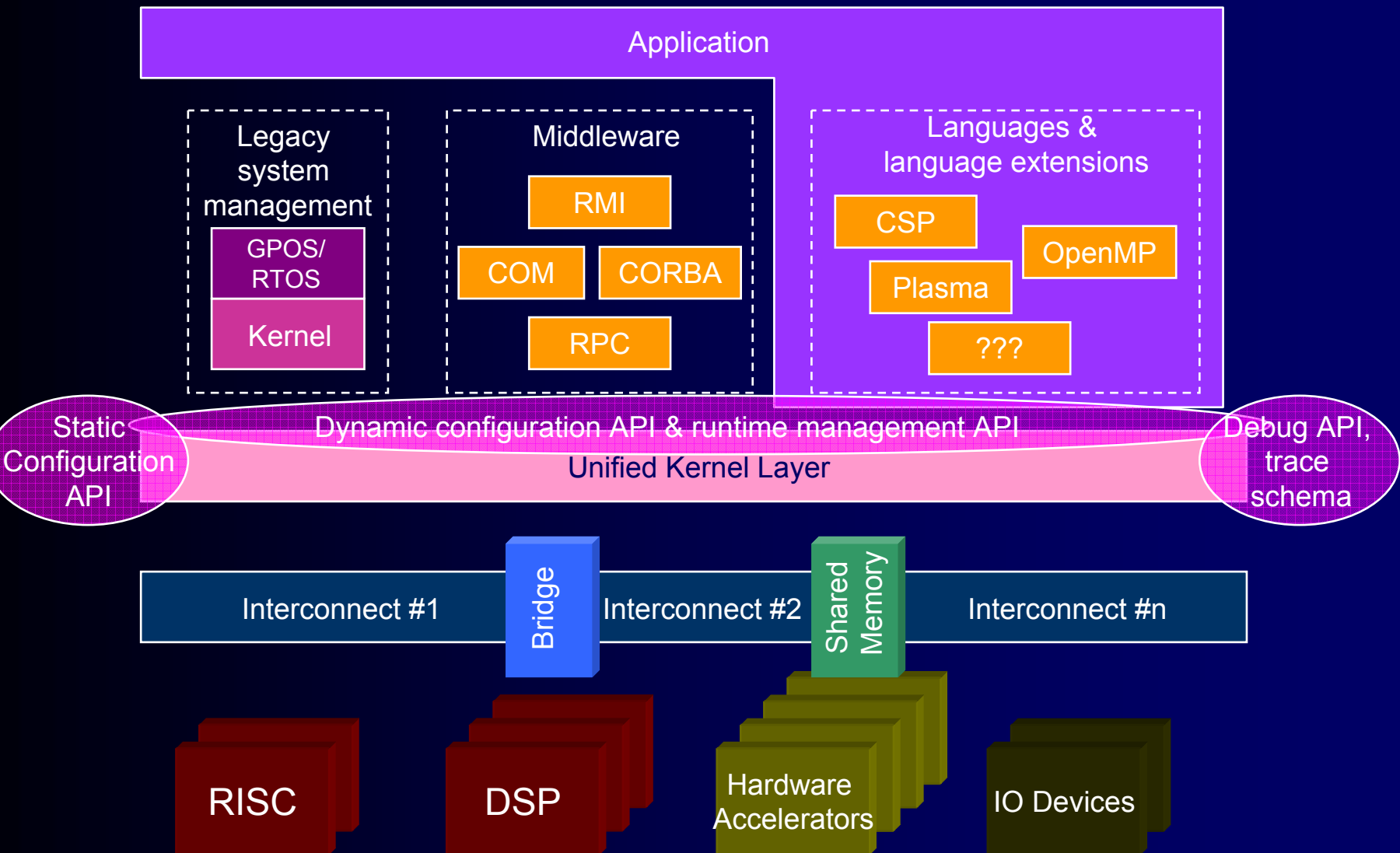
Source: ANSA (1989) Reference manual, Architecture Project Management. Camb. UK  
International Standard on Open Distributed Processing (ODP) [ISO/IEC, 1996]

Diagram: Engineering distributed objects, Wolfgang Emerick.

# Agenda

- Existing trends and techniques
- The Unified Kernel Layer
- Example: RPC over UKL
- Performance
- Conclusions

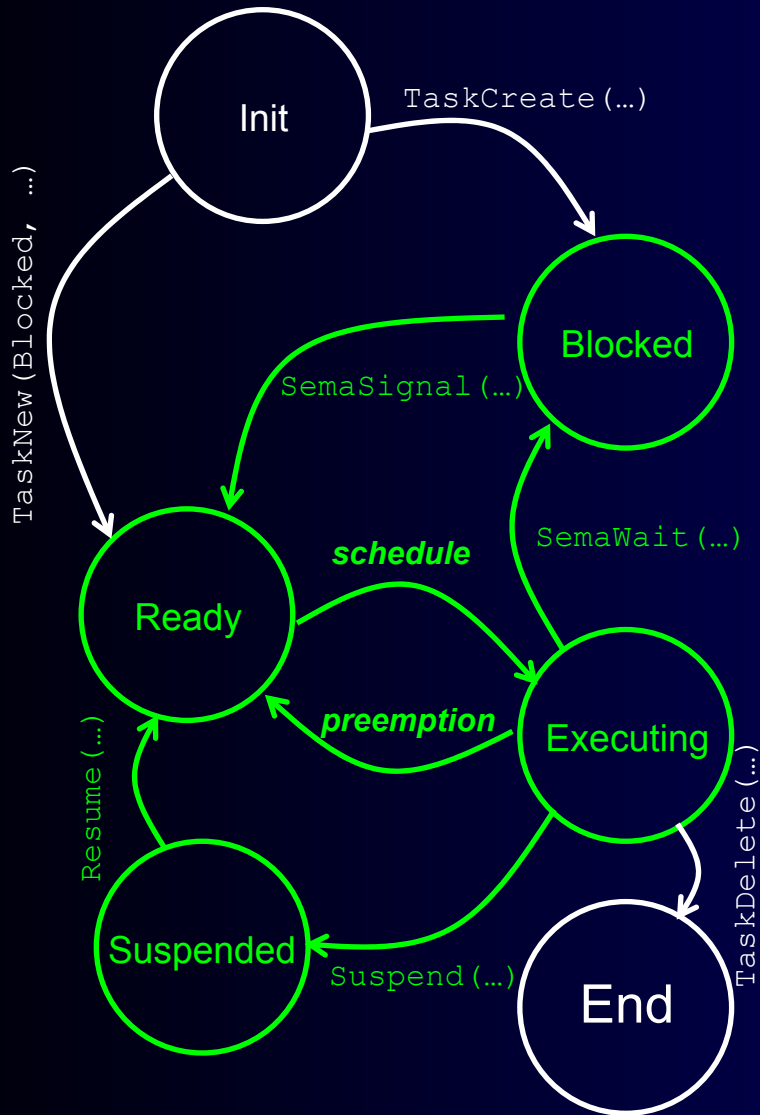
# Unified Kernel Layer



# What is the UKL?

- *The lowest common denominator, supporting...*
  - Software
    - Existing OS
    - Existing middleware
    - Existing and emerging languages (plus extensions)
  - Hardware
    - Accelerators
    - SMP & NUMA
    - Heterogeneous micro-architectures
    - Heterogeneous interconnect
- Programming model
  - Explicit thread based parallelism
    - Memory architecture agnostic
  - Fork/join model
  - Natively event based
    - Potential for low power operation
  - *No compiler directives*
- Control plane only
  - *No data plane*
  - Task state management
    - Creation/deletion
    - Suspension/resumption
    - Synchronisation
  - Task scheduling
    - Processing resource class based, enabling...
      - Dynamic load balancing
      - Dynamic power management
    - Enabling static and dynamic logical reconfiguration
- Task based debug
  - A definition of task based trace
  - Task based breakpoint/watchpoint sequences
- *Non proprietary!*

# Unified Kernel Layer runtime API



## ● Create...

- `TaskNew(...)`
  - Blocked and unblocked

## ● Manipulate

- `Suspend()`
- `Resume()`

## ● Synchronise

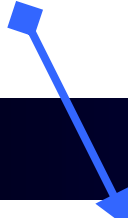
- `SemaInit()`
- `SemaDelete()`
- `SemaSignal()`
- `SemaWait()`

## ● Delete

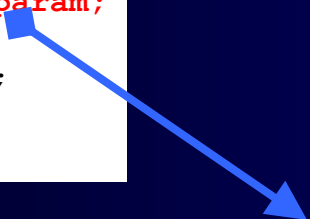
- `TaskDelete(...)`

# Scheduling with POSIX pThreads

```
int pthread_create(  
    pthread_t *tid,  
    const pthread_attr_t *attr,  
    void *(*start)(void *),  
    void *arg);
```



```
struct pthread_attr_t {  
    void *stackaddr;  
    size_t stacksize;  
    int detachstate;  
    int schedpolicy;  
    struct sched_param param;  
    int inheritsched;  
    int contentionscope;  
};
```



- Policy oriented definition
- *Implicit* scheduler structure definition
  - Implies processing resource homogeneity

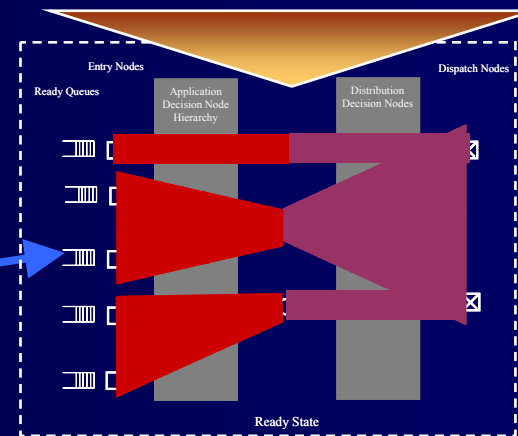
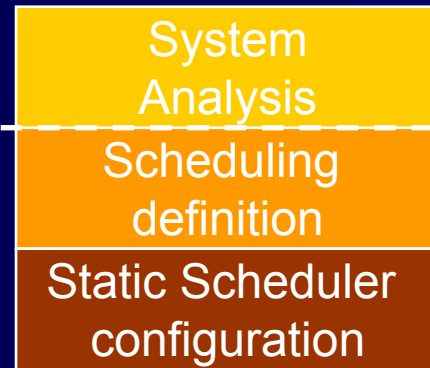
FIFO: SCHED\_FIFO  
Round robin: SCHED\_RR  
???: SCHED\_OTHER



# Scheduling with UKL threads

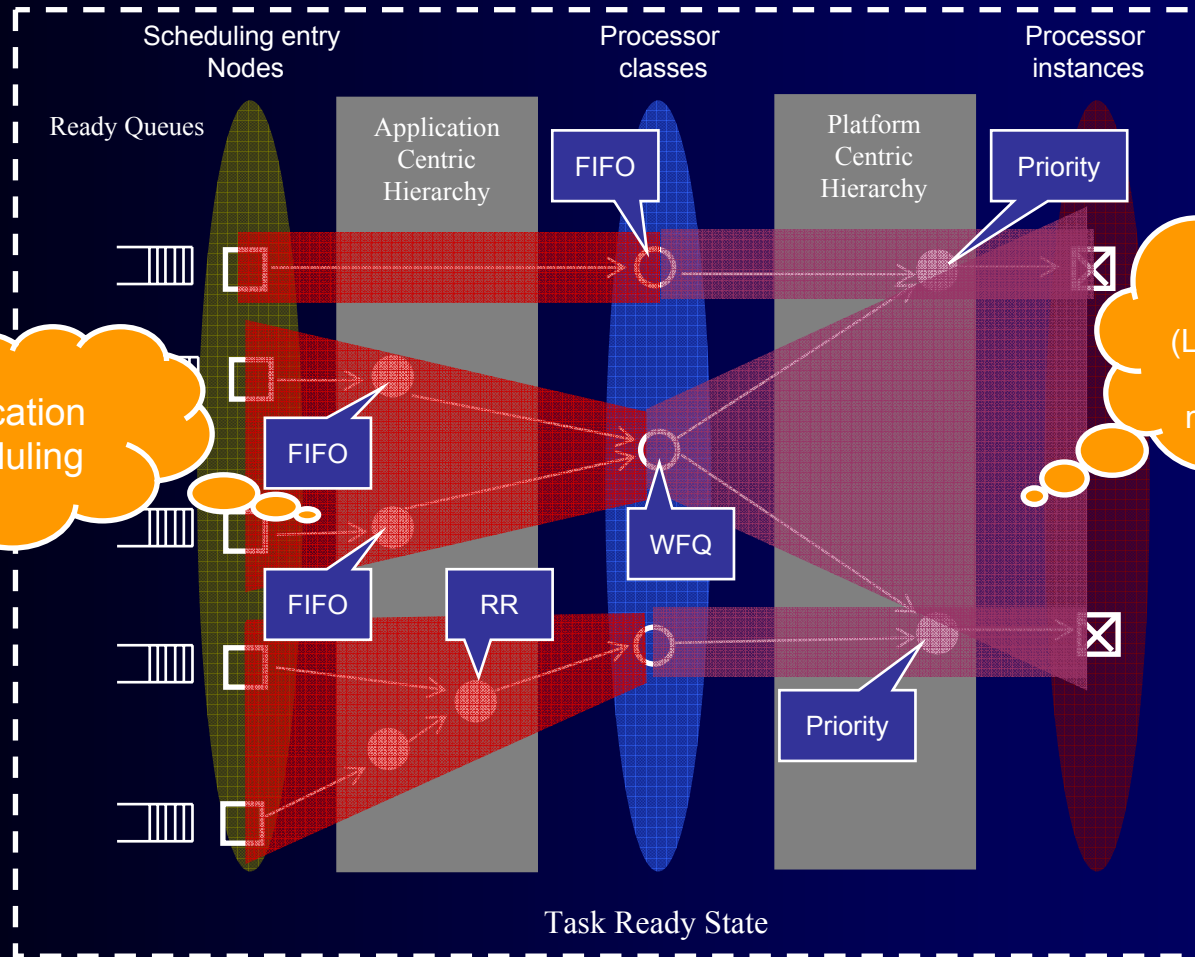
```
SyWReturn_t SyWTaskNew(  
    SyWTCB_t *pTask,  
    BOOLEAN IsBlocked);
```

```
struct SyWTCB  
{  
    U32 (*fpTask)(void *);  
    U32 *pParams;  
    U32 Priority; //Metric0  
    U32 Metric1;  
    U16 SchedulingNode;  
    // Implementation  
    // specific params  
    ...  
};
```

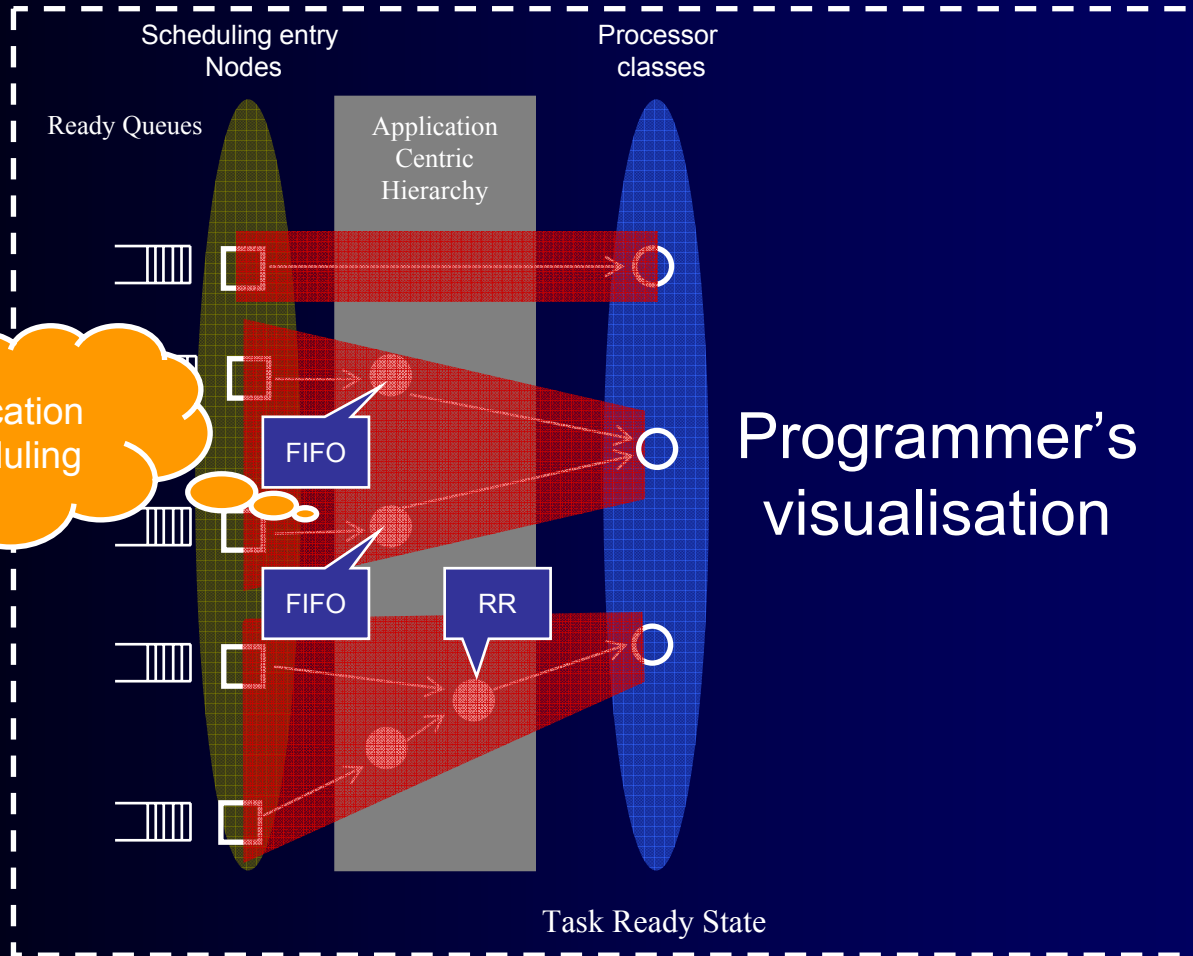


- Hierarchy oriented definition.
- *Explicit* scheduler structure

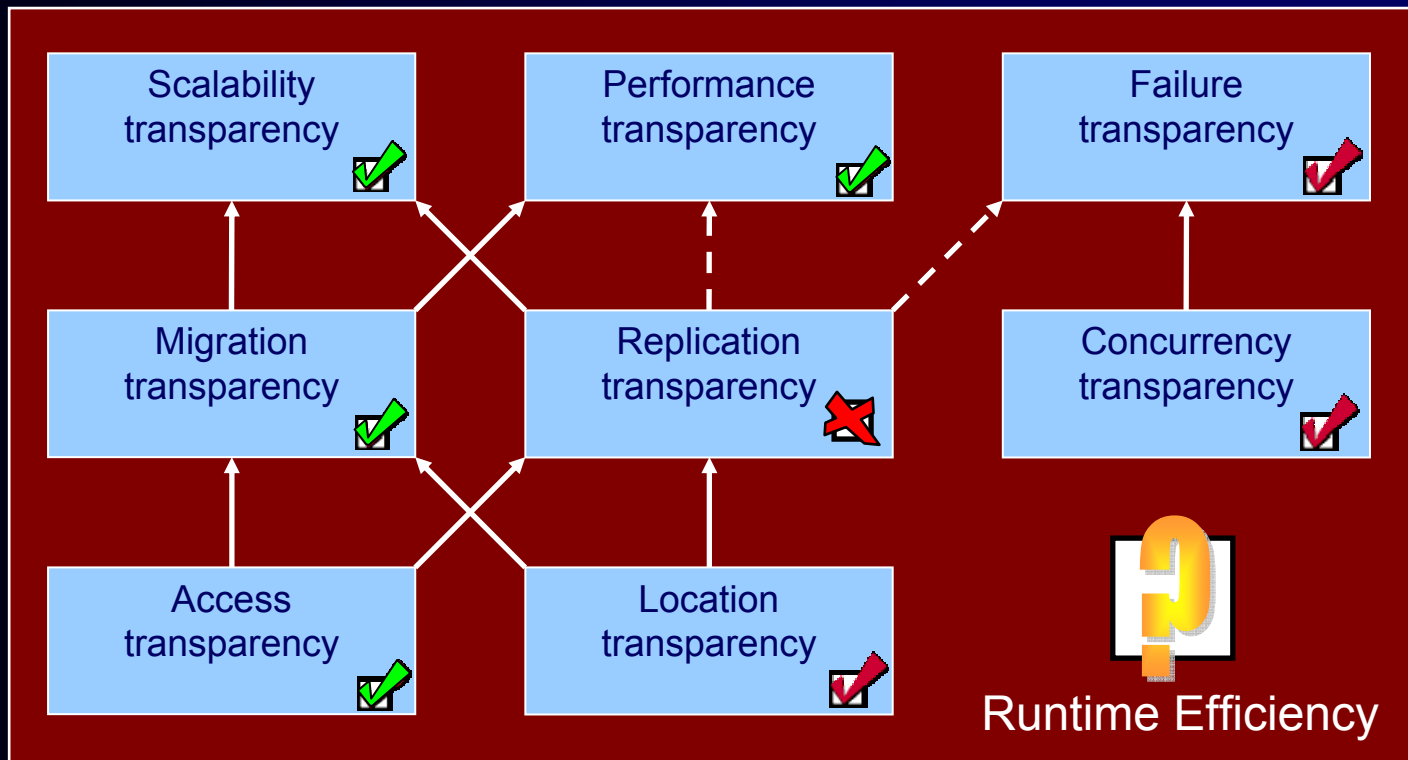
# Configuration Diagram - Example



# Configuration Diagram - Example



# UKL transparency



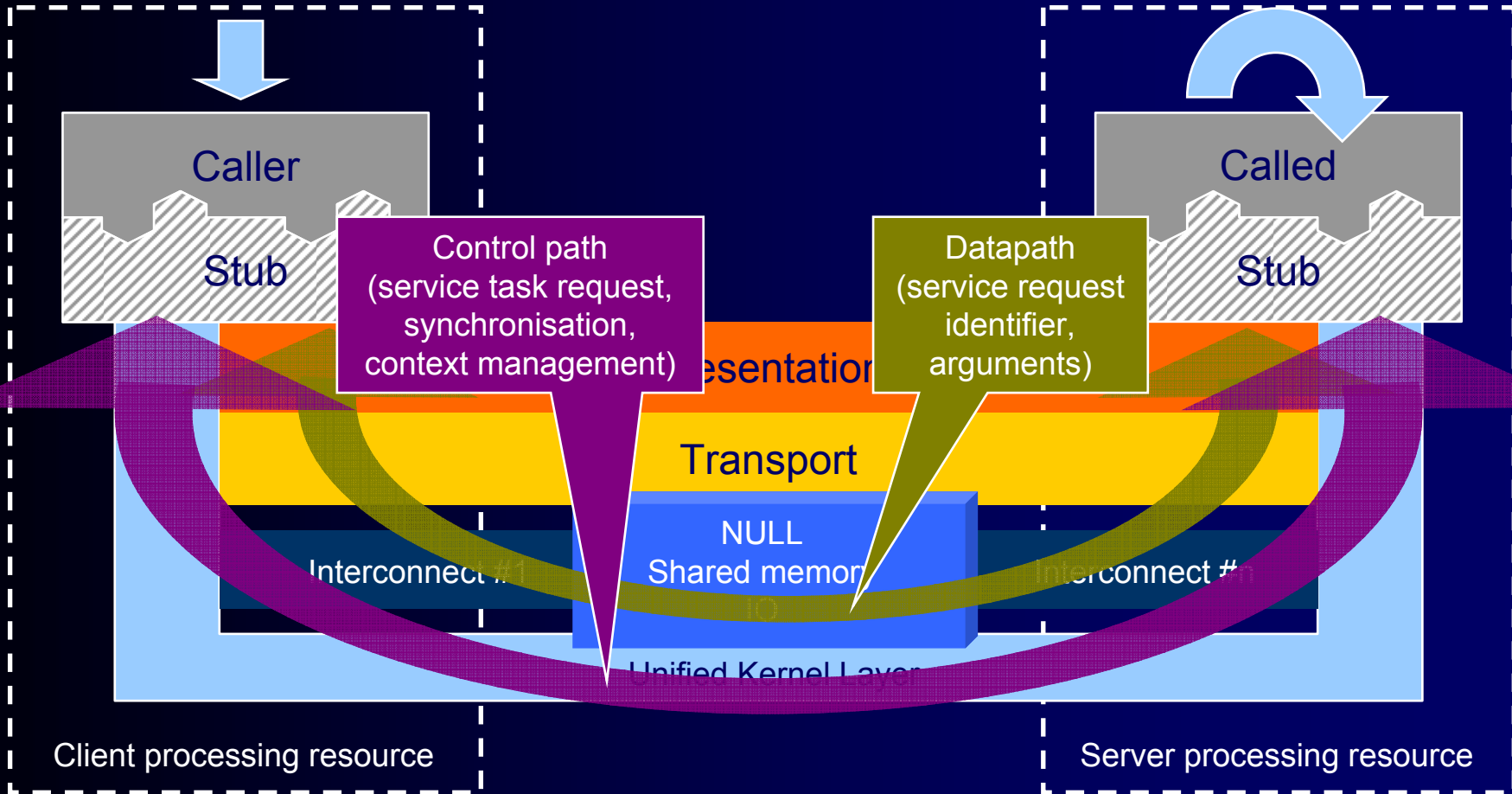
- ✓ Strong support
- ✓ Partial support
- 🔍 Possible (implementation dependant)
- ✗ No support

# Agenda

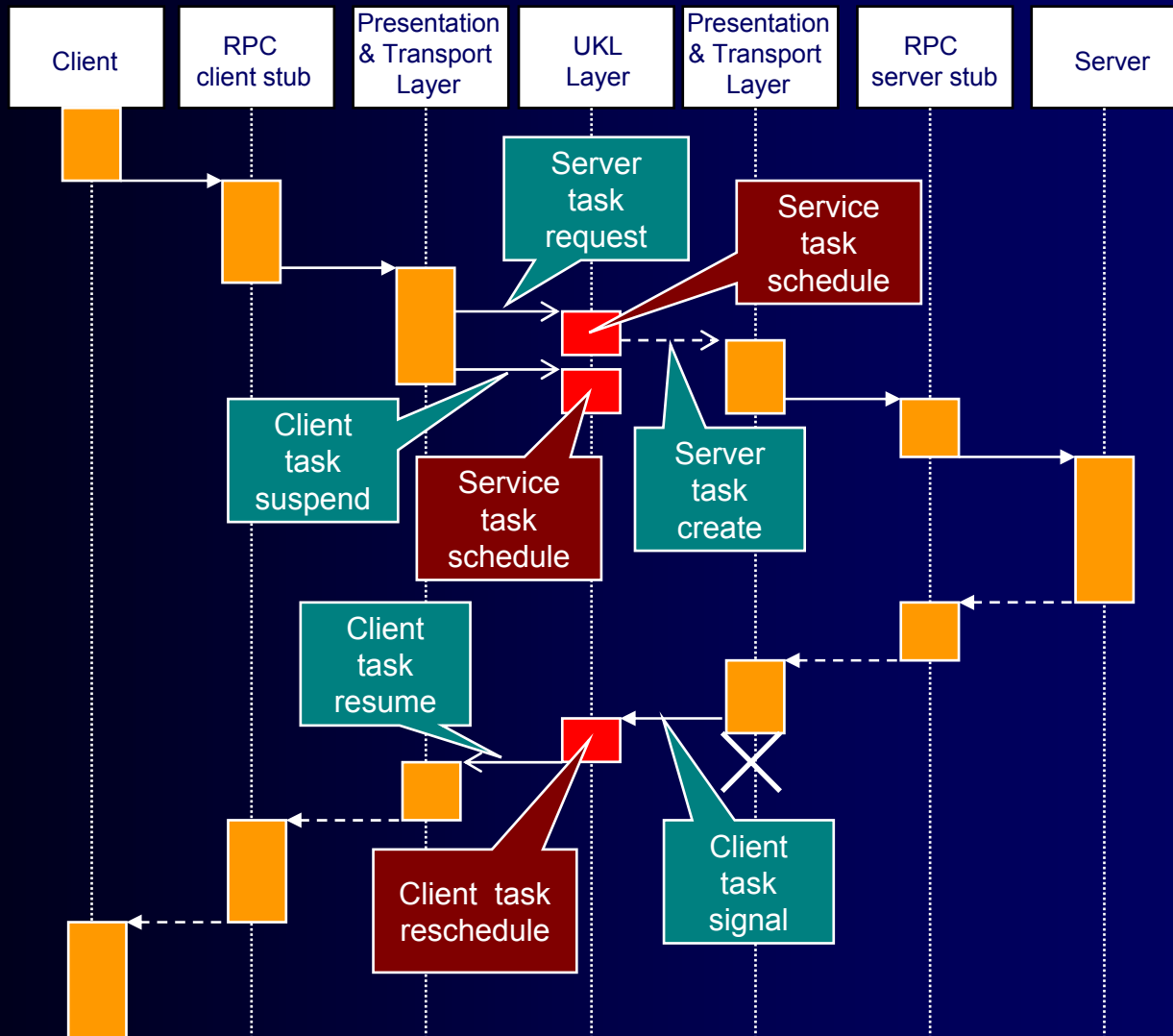
- Existing trends and techniques
- The Unified Kernel Layer
- Example: RPC over UKL
- Performance
- Conclusions

# RPC over UKL

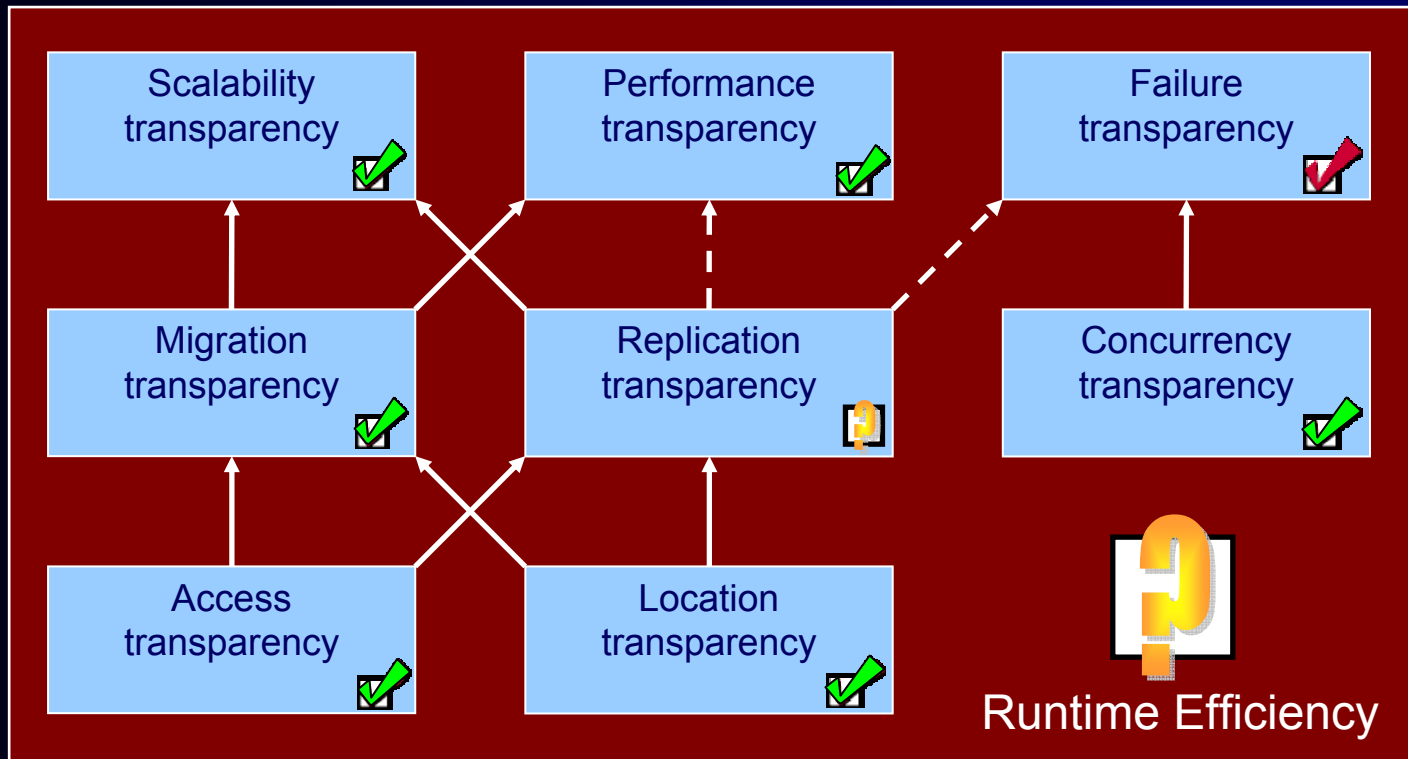
RPCall (Args, \*pReturnArgs)





# Client server sequence diagram



# RPC over UKL transparency



-  Strong support
-  Partial support
-  Possible (implementation dependant)
-  No support

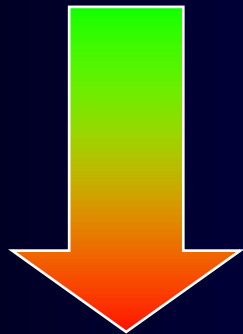


# Agenda

- Existing trends and techniques
- The Unified Kernel Layer
- Example: RPC over UKL
- Performance
- Conclusions

# Resource management performance

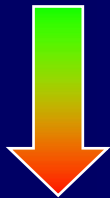
Resource management performance  $\alpha$



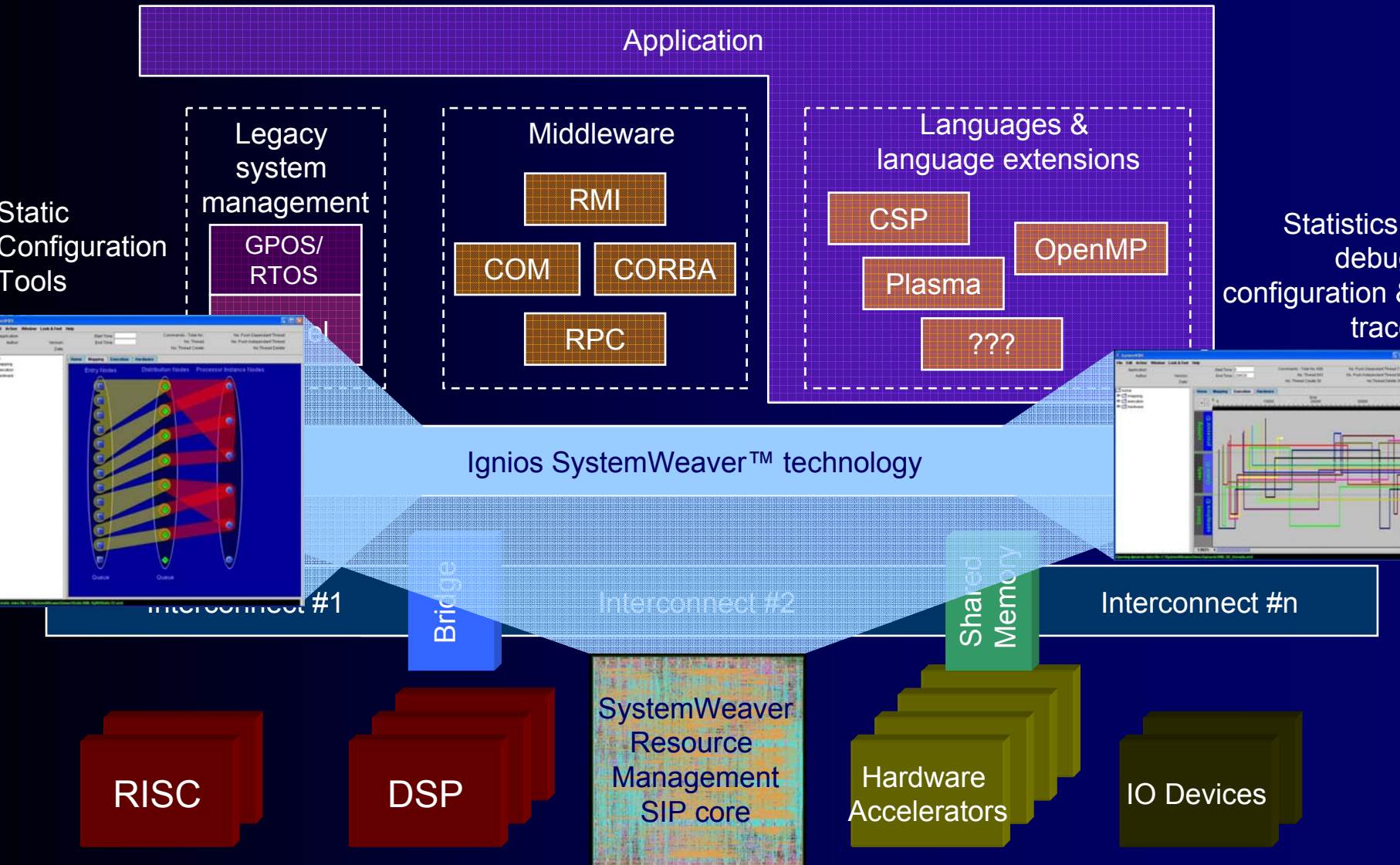
$$\frac{1}{\text{Event frequency}}$$
$$\frac{1}{\text{System complexity}}$$



Task granularity



# SystemWeaver technology



# Agenda

- Existing trends and techniques
- The Unified Kernel Layer
- Example: RPC over UKL
- Performance
- Conclusions

# Conclusions

- UKL abstraction
  - Lightweight
  - Class based programming model
    - Scalable applications
    - Dynamically scalable power consumption
  - Event driven
    - Low power
  - Based on existing thread abstractions
    - With enhanced scheduling capabilities
  - Makes debug challenging
- Must retain flexibility
  - Static and dynamic reconfigurability of control path
  - Must inter-operate with existing OS technology
- Should be open standard
  - Judge implementations on their merits
- Must be efficient at runtime
  - Hardware support required

Thank-you