Meeting the Challenges of Ultra-Large-Scale Systems

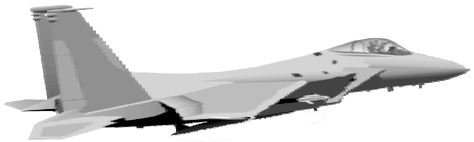
Tuesday, July 11, 2006, OMG RTWS, Arlington, VA

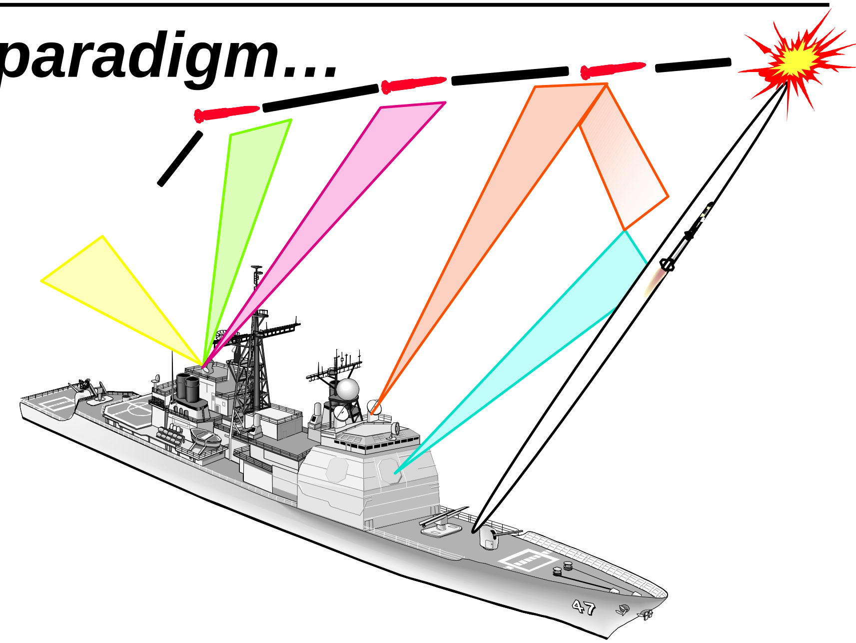
Dr. Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

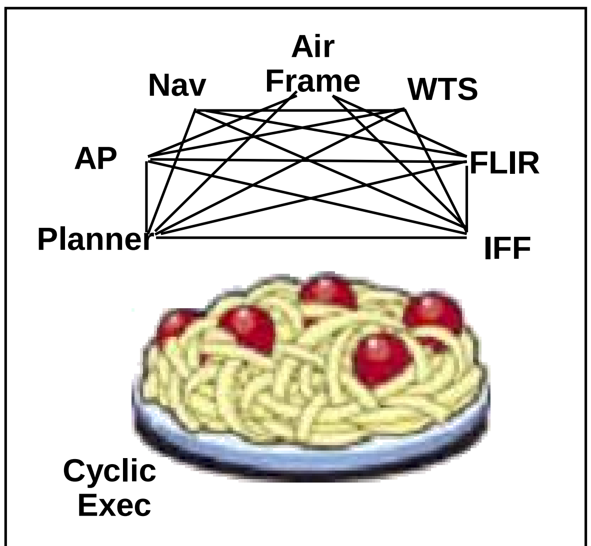
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee

Prior R&D Progress

From this design paradigm…







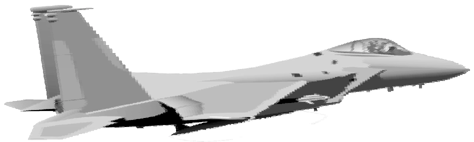
The designs of legacy distributed real-time & embedded (DRE) systems tend to be:

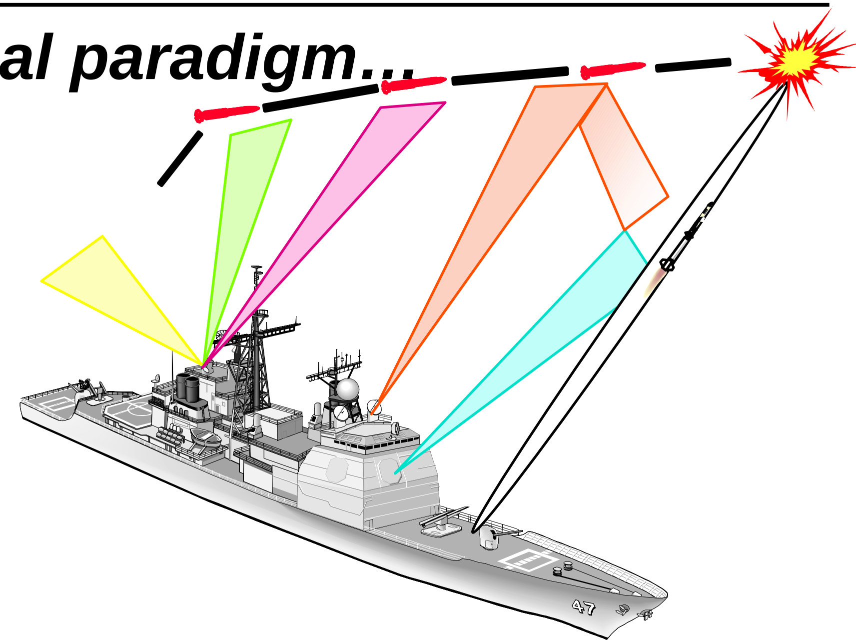
- Stovepiped
- Proprietary
- Brittle & non-adaptive
- Expensive to develop & evolve
- Vulnerable

Problem: Small changes can break nearly anything & everything

Prior R&D Progress

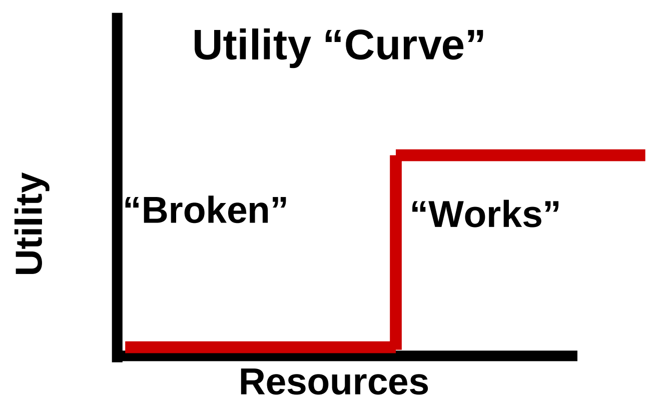
…and this operational paradigm…





Real-time QoS requirements for traditional DRE systems:

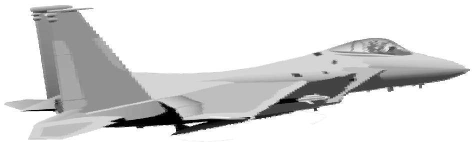
- Ensure end-to-end QoS, e.g.,
  - Minimize latency, jitter, & footprint
  - Bound priority inversions
- Allocate & manage resources *statically*

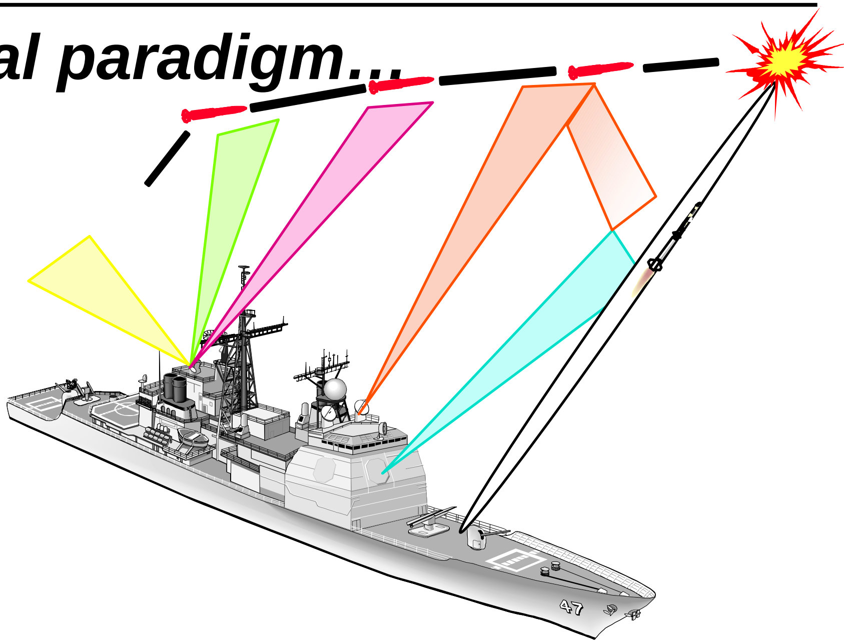


"Hard" Requirements

Prior R&D Progress

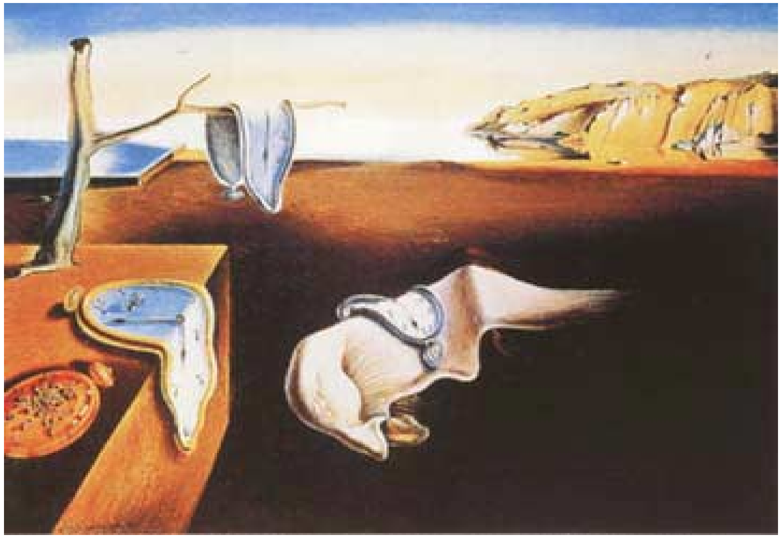
…and this operational paradigm…





Real-time QoS requirements for traditional DRE systems:

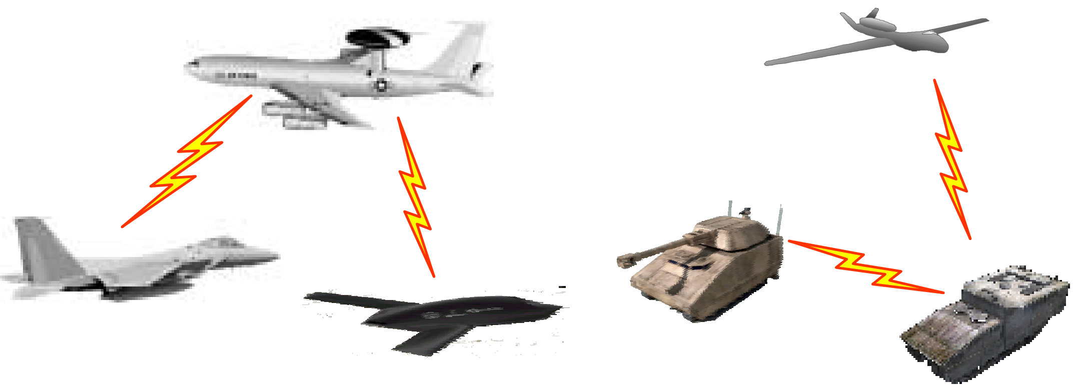
- Ensure end-to-end QoS, e.g.,
  - Minimize latency, jitter, & footprint
  - Bound priority inversions
- Allocate & manage resources *statically*

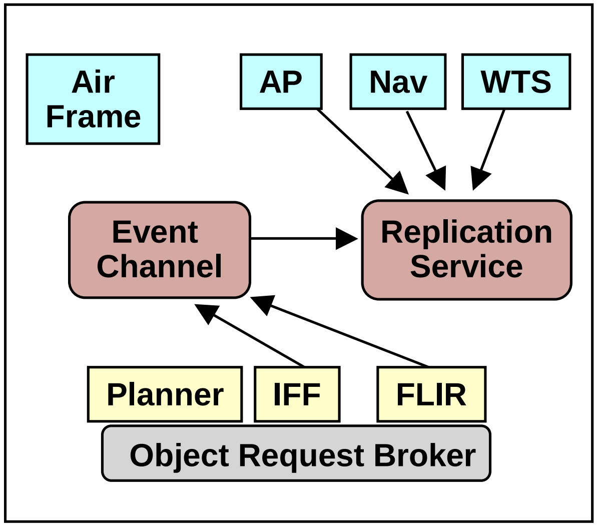


Problem: Lack of any resource can break nearly everything

Prior R&D Progress

…to this design paradigm…





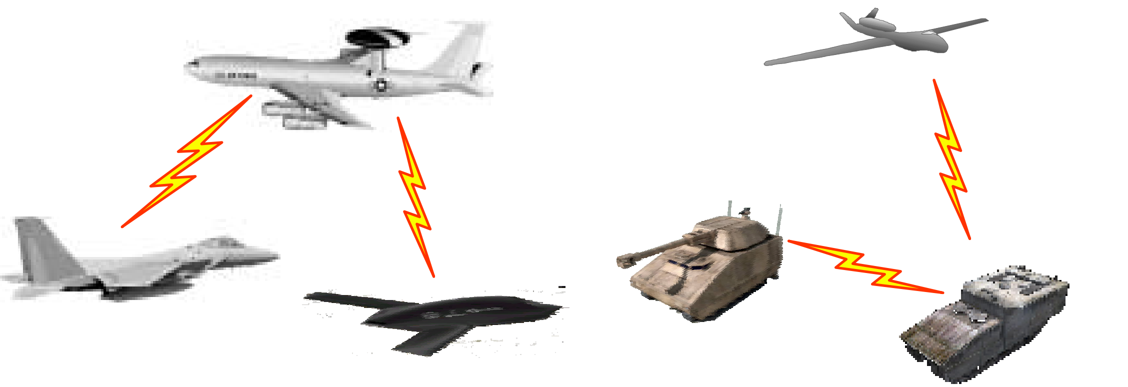
The designs of today's leading-edge DRE systems tend to be more:

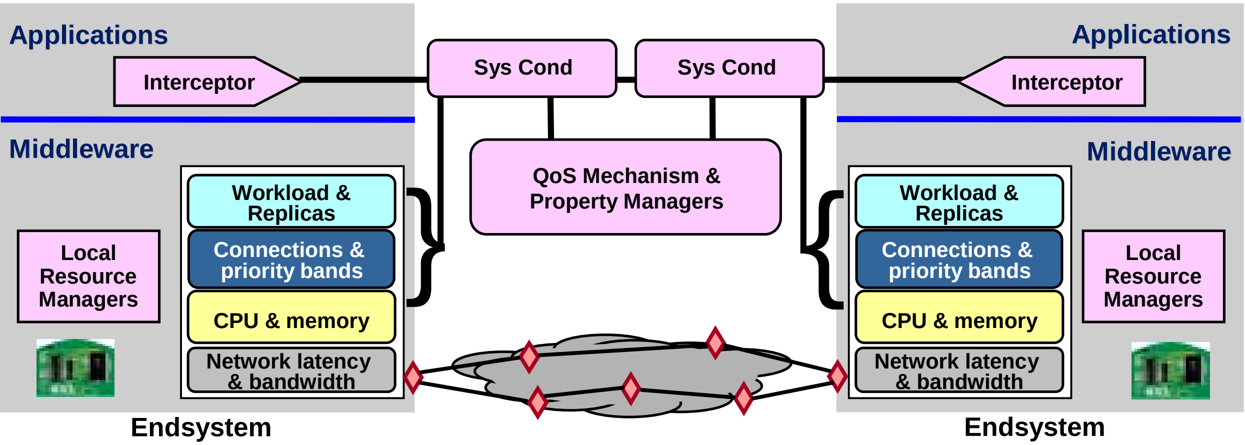
- Layered & componentized
- Standard & COTS
- Robust to failures & adaptive to operating conditions
- Cost effective to evolve & retarget

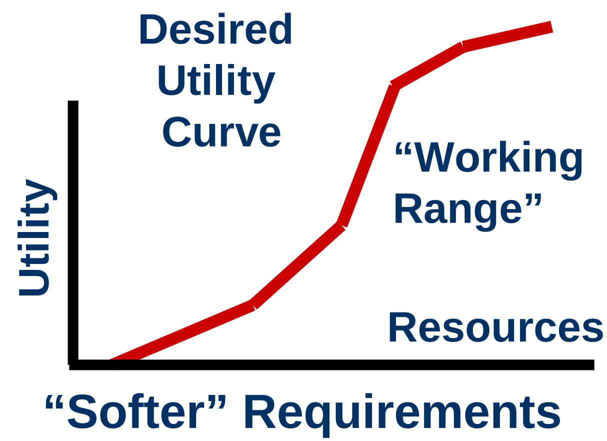
Result: new requirements & design changes can be handled more flexibly

Prior R&D Progress

…and this operational paradigm…



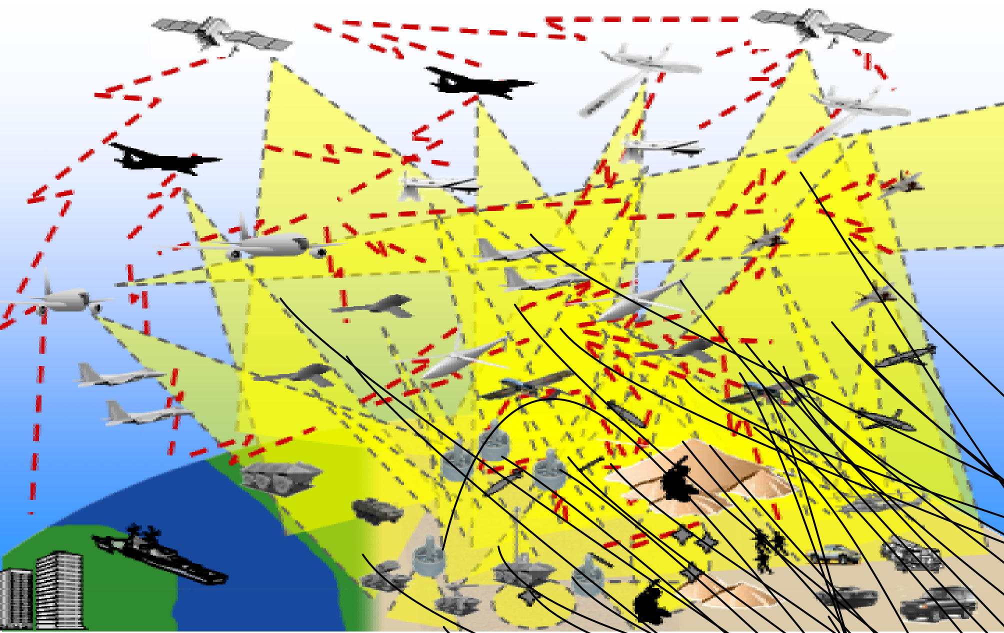




Adaptive QoS capabilities at multiple network, OS, & middleware layers

Result: better support for network-centric operations with scarce resources

New Challenge: Ultra-Large-Scale (ULS) Systems

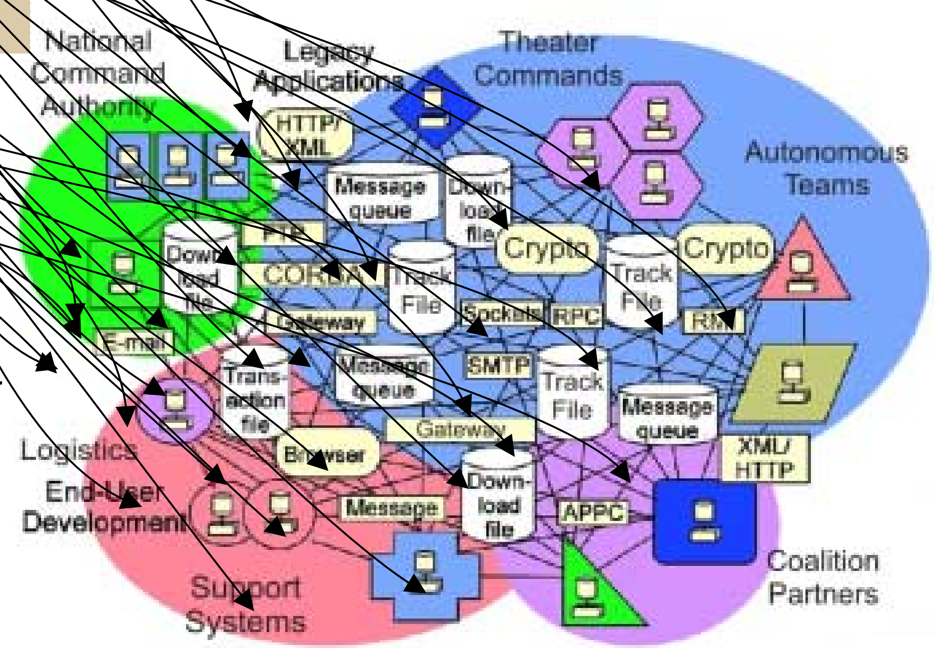


Key ULS ***problem space*** challenges

- Highly dynamic & distributed development & operational environments
- Stringent simultaneous quality of service (QoS) demands
- Very diverse & complex network-centric application domains

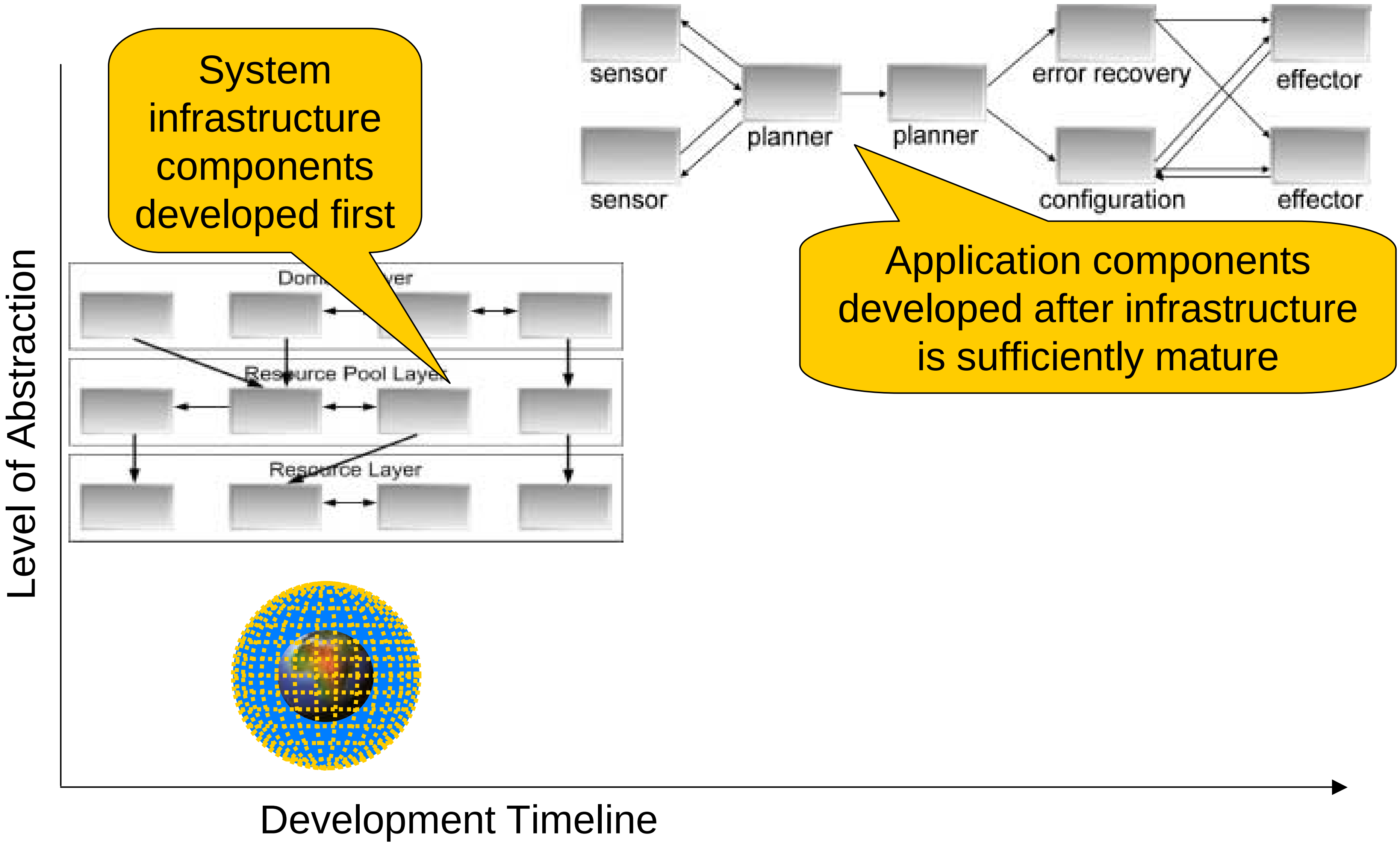
Key ULS ***solution space*** challenges

- Enormous accidental & inherent complexities
- Continuous evolution & change
- Highly heterogeneous platform, language, & tool environments

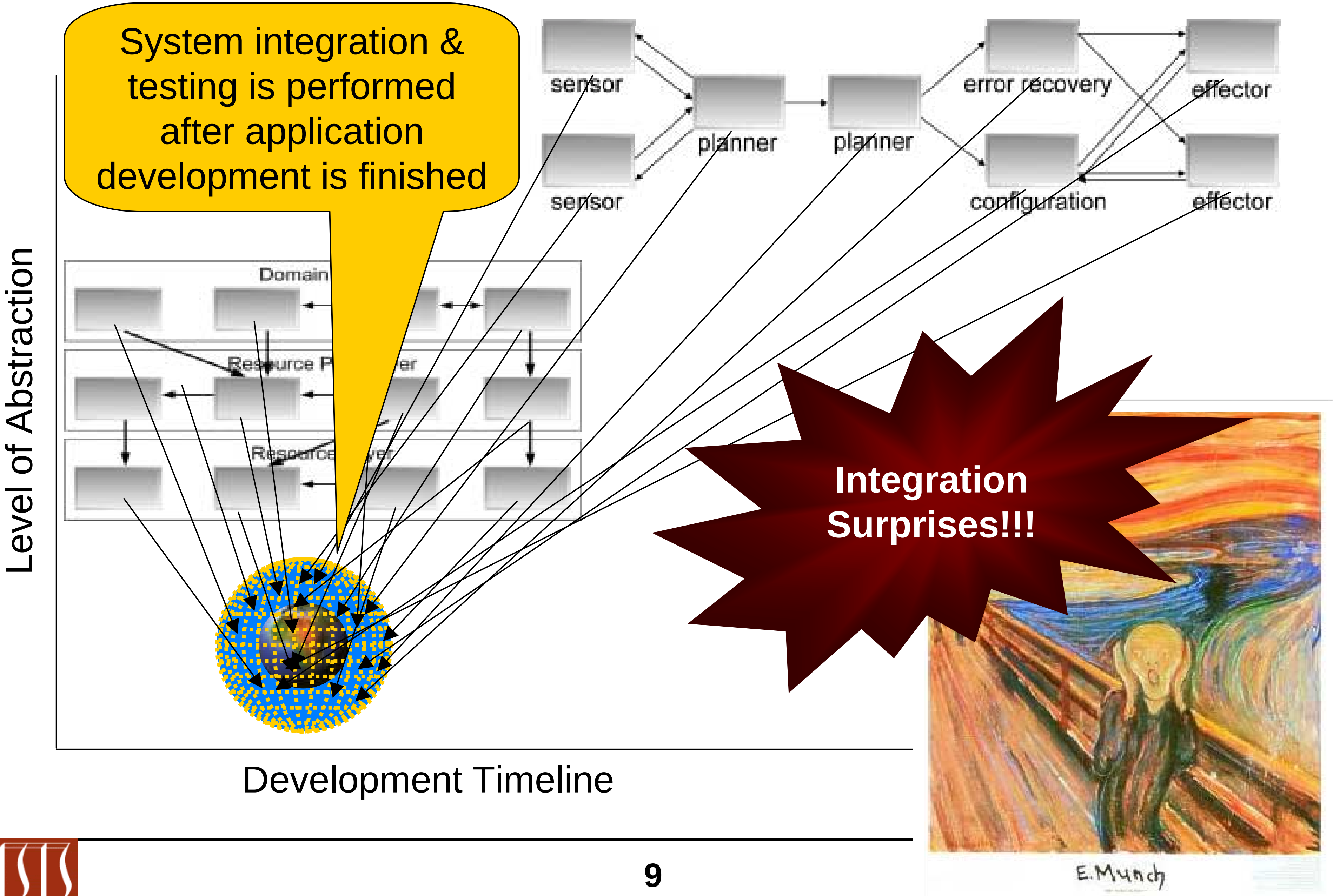


Mapping *problem space requirements* to *solution space artifacts* is very hard

Serialized Phasing is Common in ULS Systems

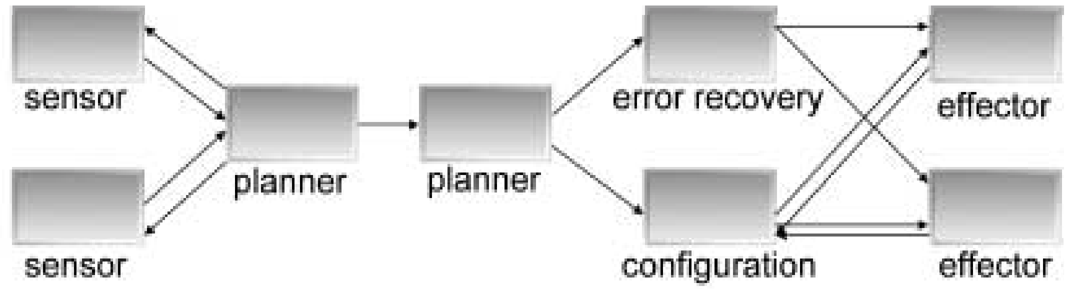


Serialized Phasing is Common in ULS Systems

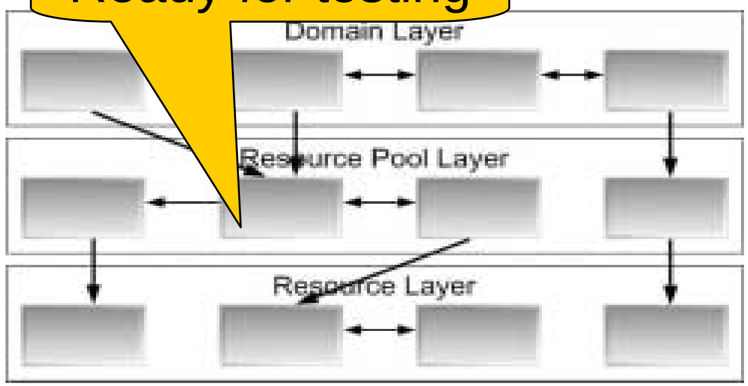


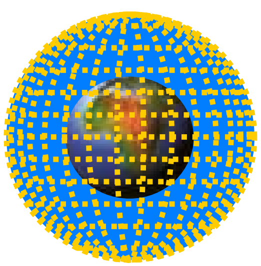
Complexities of Serialized Phasing

Still in development



Ready for testing





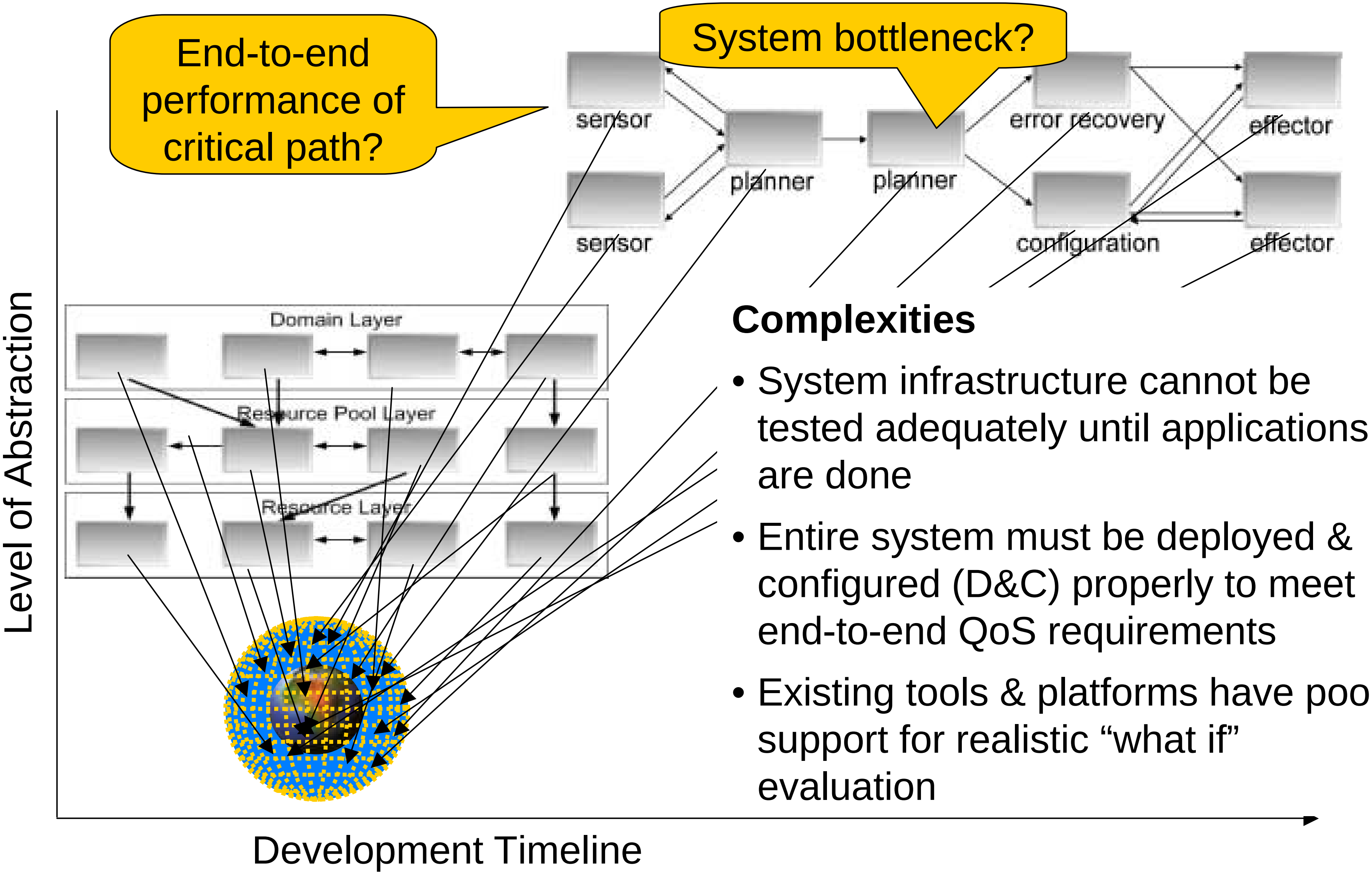
Level of Abstraction

Development Timeline

Complexities

- System infrastructure cannot be tested adequately until applications are done

Complexities of Serialized Phasing

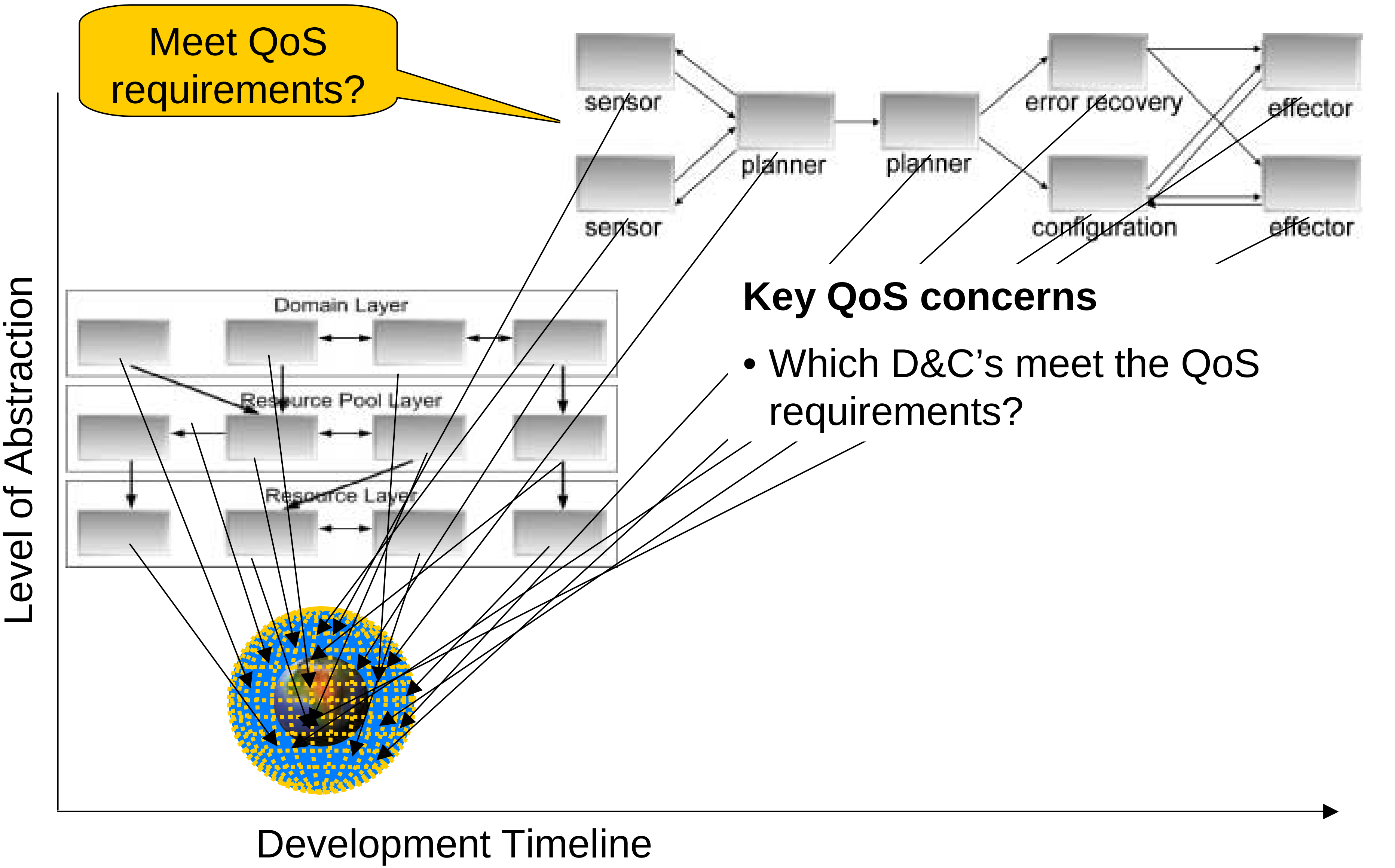


Complexities

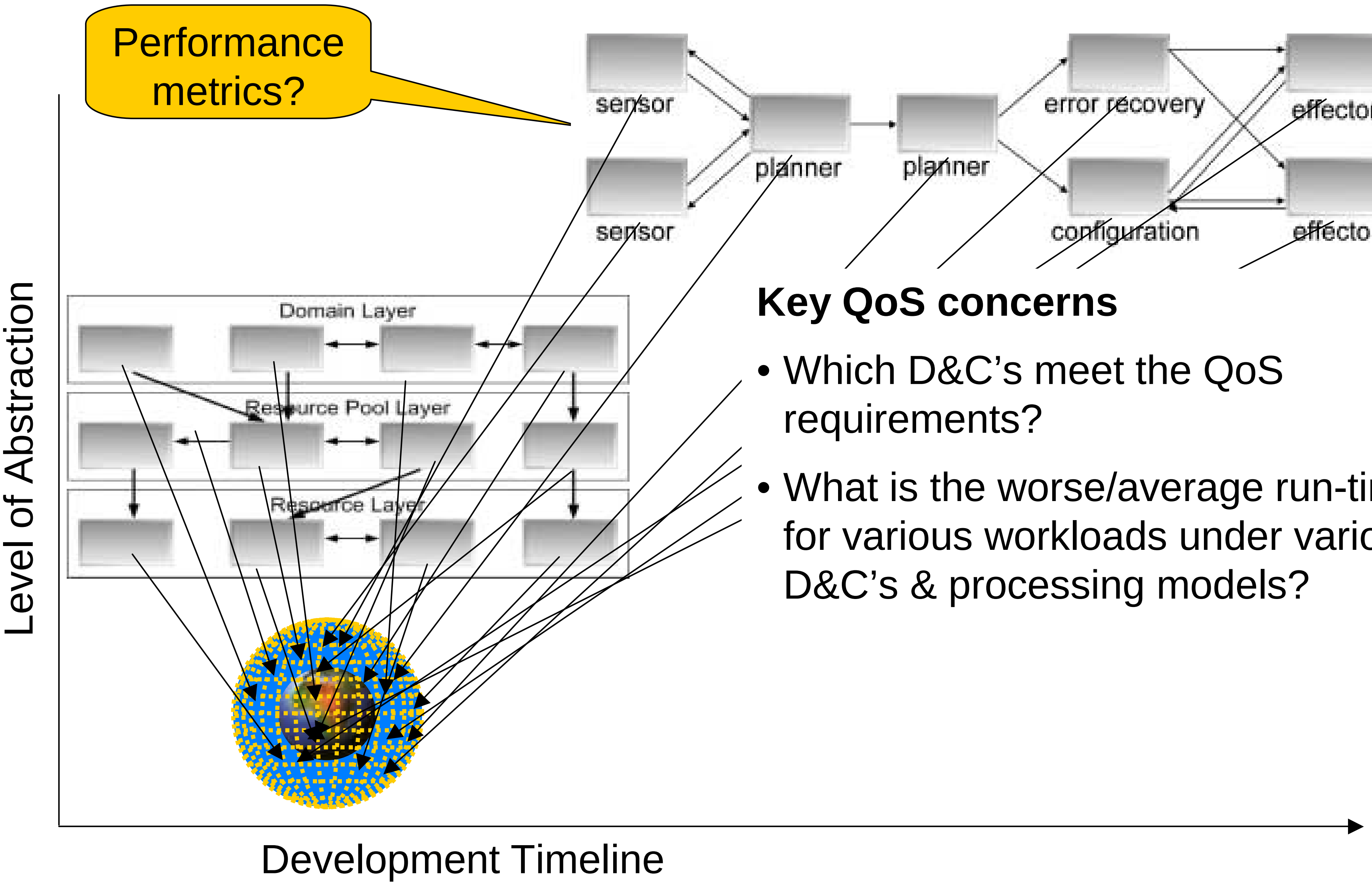
- System infrastructure cannot be tested adequately until applications are done
- Entire system must be deployed & configured (D&C) properly to meet end-to-end QoS requirements
- Existing tools & platforms have poor support for realistic "what if" evaluation

QoS requirements of components & system often unknown until late in lifecycle

Unresolved QoS Concerns with Serialized Phasing



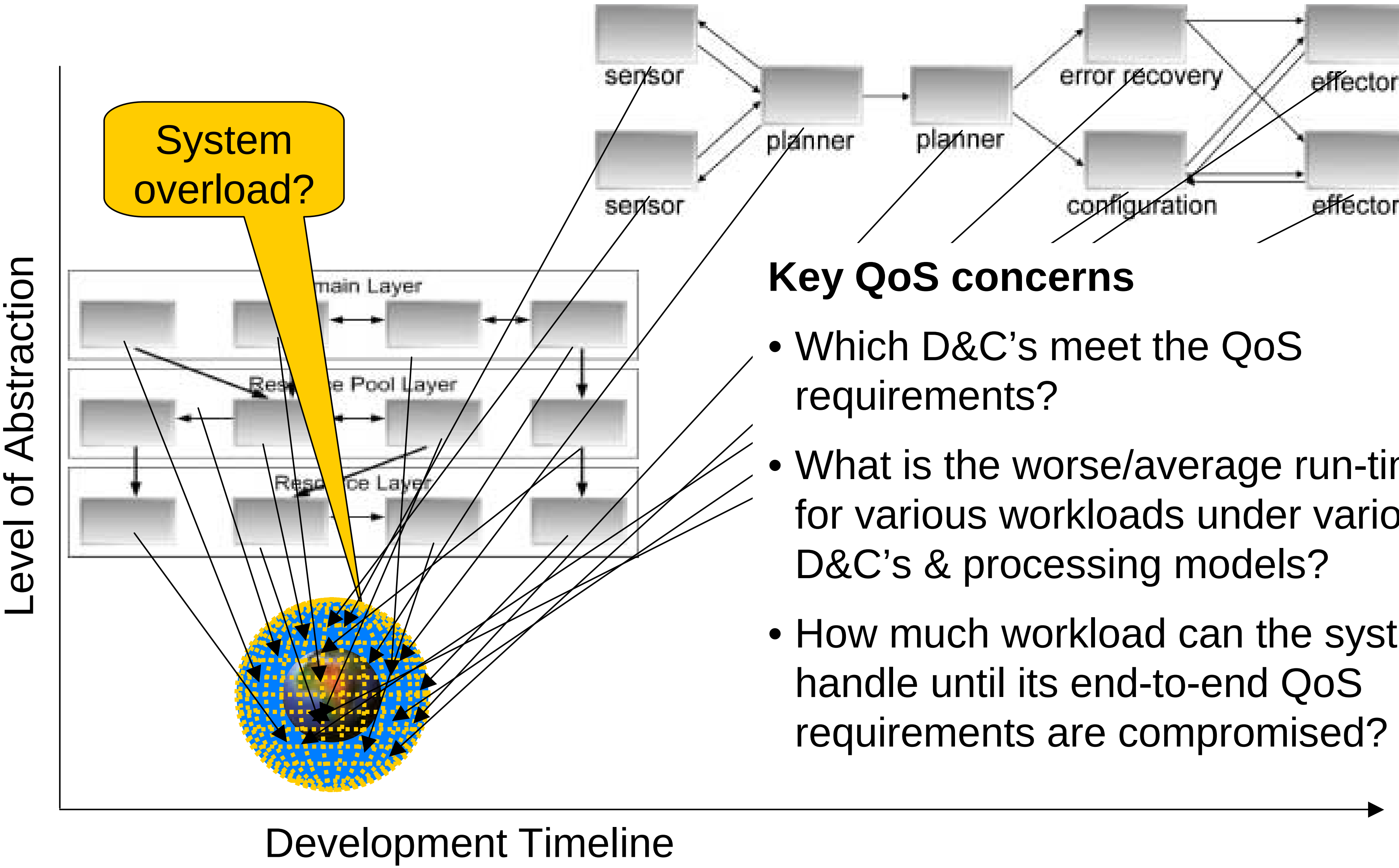
Unresolved QoS Concerns with Serialized Phasing



Key QoS concerns

- Which D&C's meet the QoS requirements?
- What is the worse/average run-time for various workloads under various D&C's & processing models?

Unresolved QoS Concerns with Serialized Phasing

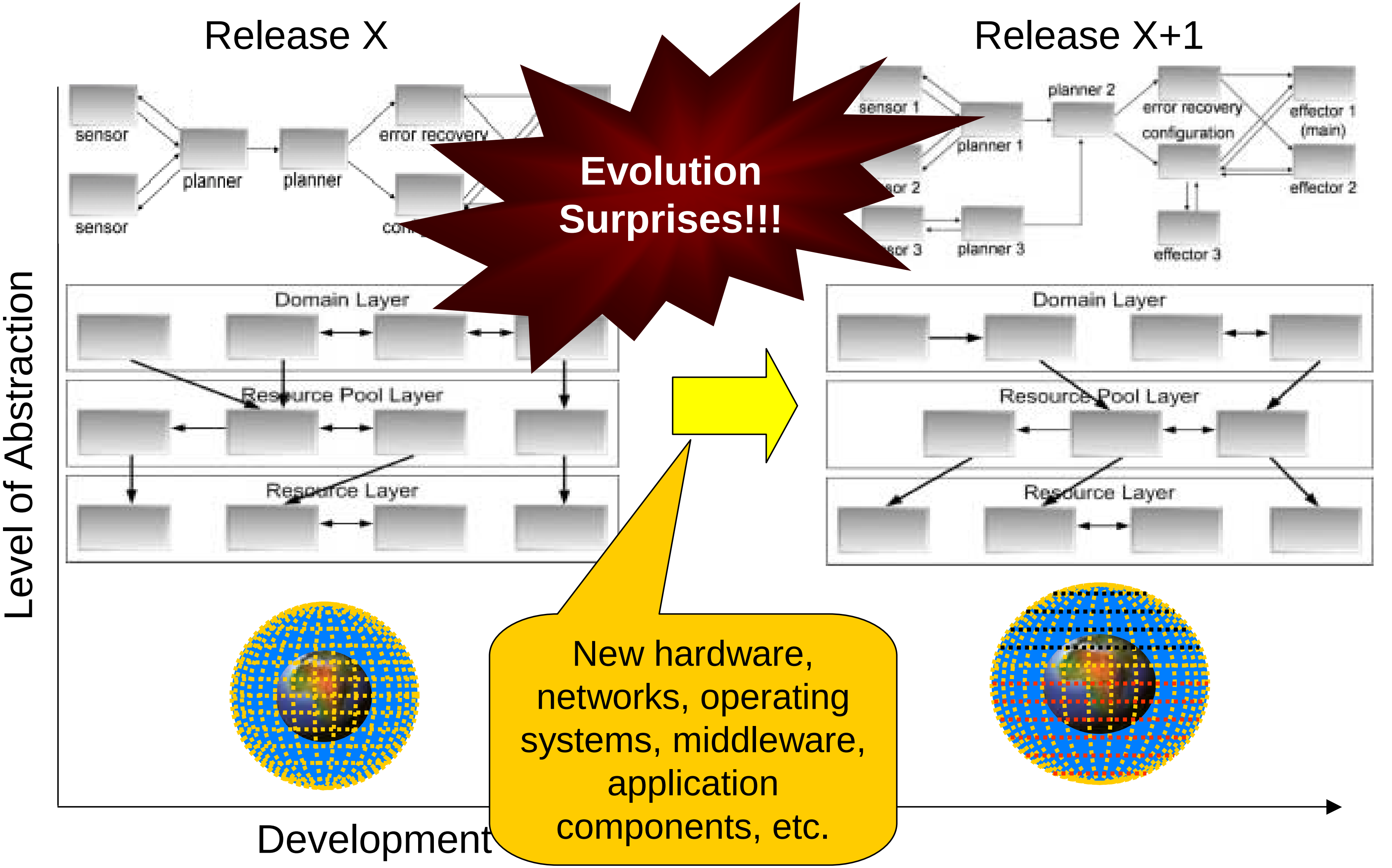


Key QoS concerns

- Which D&C's meet the QoS requirements?
- What is the worse/average run-time for various workloads under various D&C's & processing models?
- How much workload can the system handle until its end-to-end QoS requirements are compromised?

It can take a *long* time (years) to address QoS concerns with serialized phasing

Related Large-Scale System Development Problems



Promising Approach for DoD Systems Challenges:

System Execution Modeling (SEM) Tools

Tools to express & validate design rules

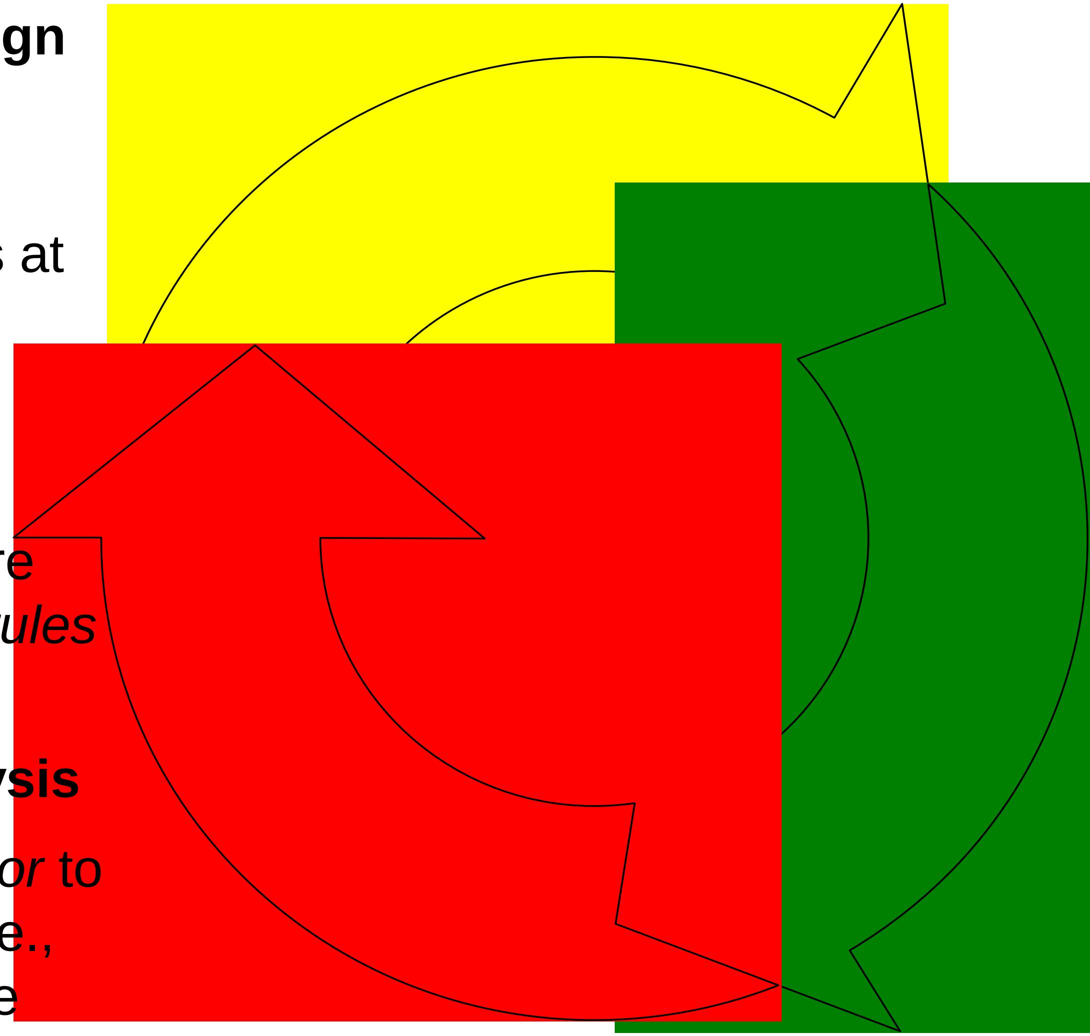
- Help applications & developers adhere to system specifications at design-time

Tools to ensure design conformance

- Help properly deploy & configure applications to enforce *design rules* throughout system lifecycle

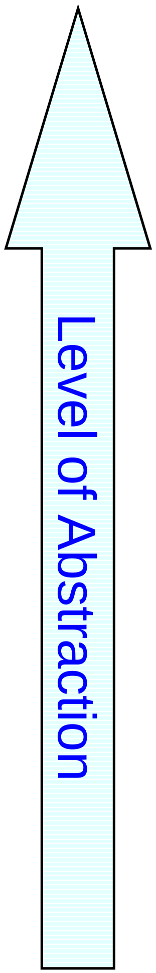
Tools to conduct "what if" analysis

- Help analyze QoS concerns *prior* to completing the entire system, i.e., before system integration phase

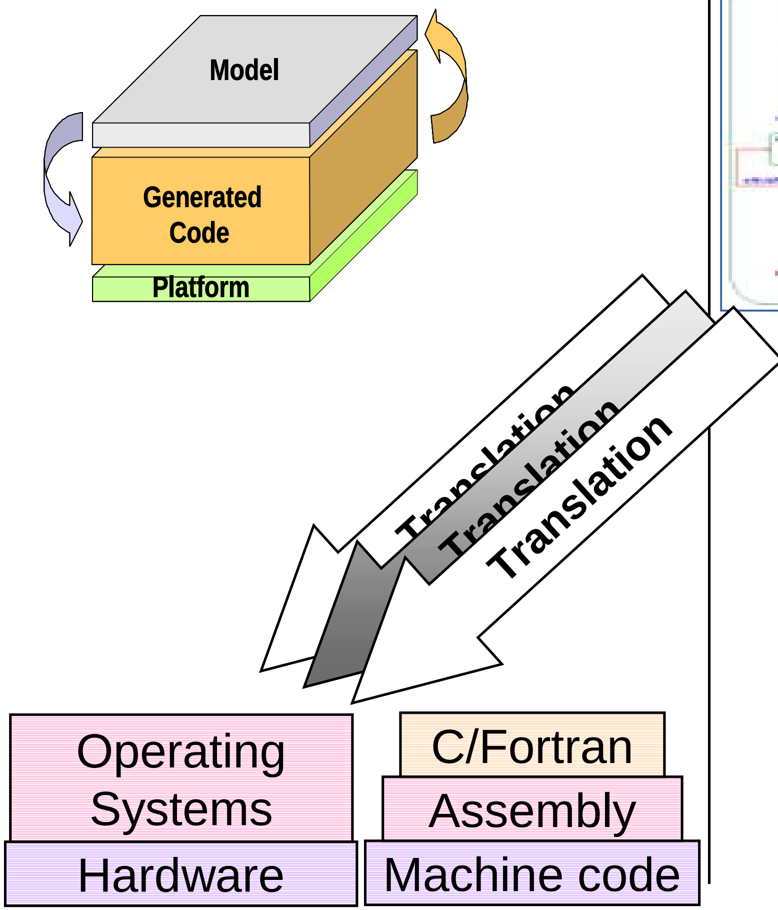


SEM tools should be applied continuously when developing software elements

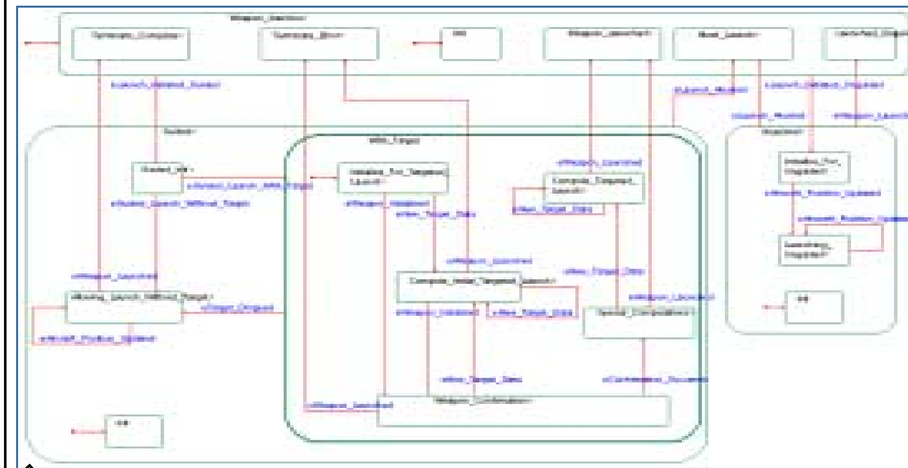
Technology Evolution (1/4)



Programming Languages & Platforms



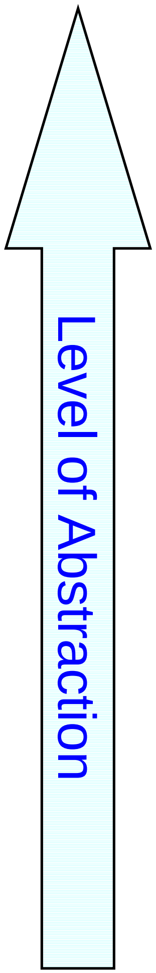
Model-Driven Engineering (MDE)



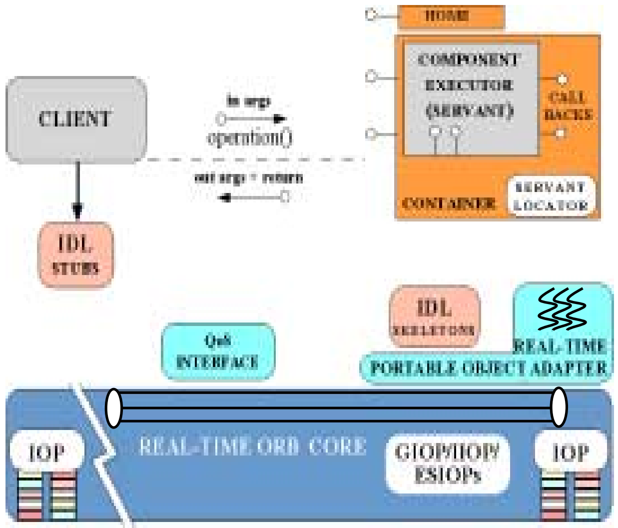
- State chart
- Data & process flow
- Petri Nets

Large Semantic Gap

Technology Evolution (2/4)



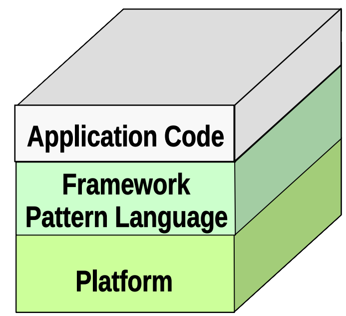
Programming Languages & Platforms



Components
Frameworks
Class Libraries
Operating Systems
Hardware

C++/Java
C/Fortran
Assembly
Machine code

- Newer 3rd-generation languages & platforms have raised abstraction level significantly
  - “Horizontal” platform reuse alleviates the need to redevelop common services



- There are two problems, however:
  - Platform complexity evolved faster than 3rd-generation languages
  - Much application/platform code still (unnecessarily) written manually

Technology Evolution (3/4)

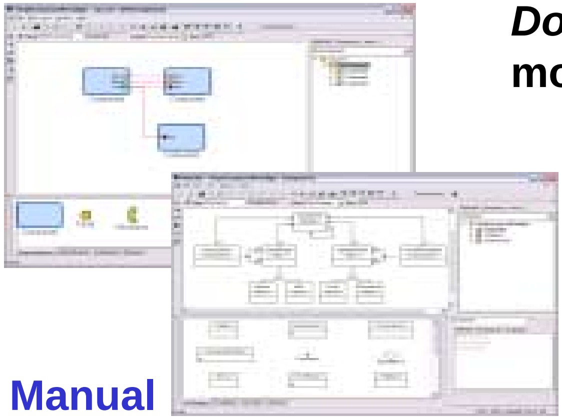
Level of Abstraction

Programming Languages & Platforms

Saturation!!!!

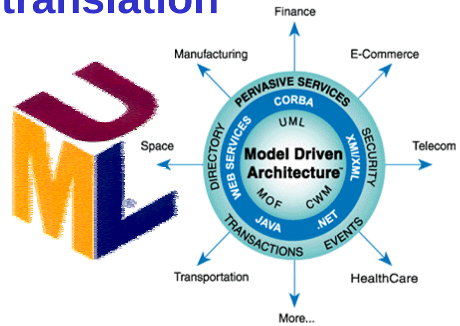
Components	
Frameworks	
Class Libraries	C++/Java
Operating Systems	C/Fortran
	Assembly
Hardware	Machine code

Model-Driven Engineering (MDE)



Manual translation

***Domain-specific* modeling languages**
- ESML
- PICML
- Mathematica
- Excel
- *Metamodels*



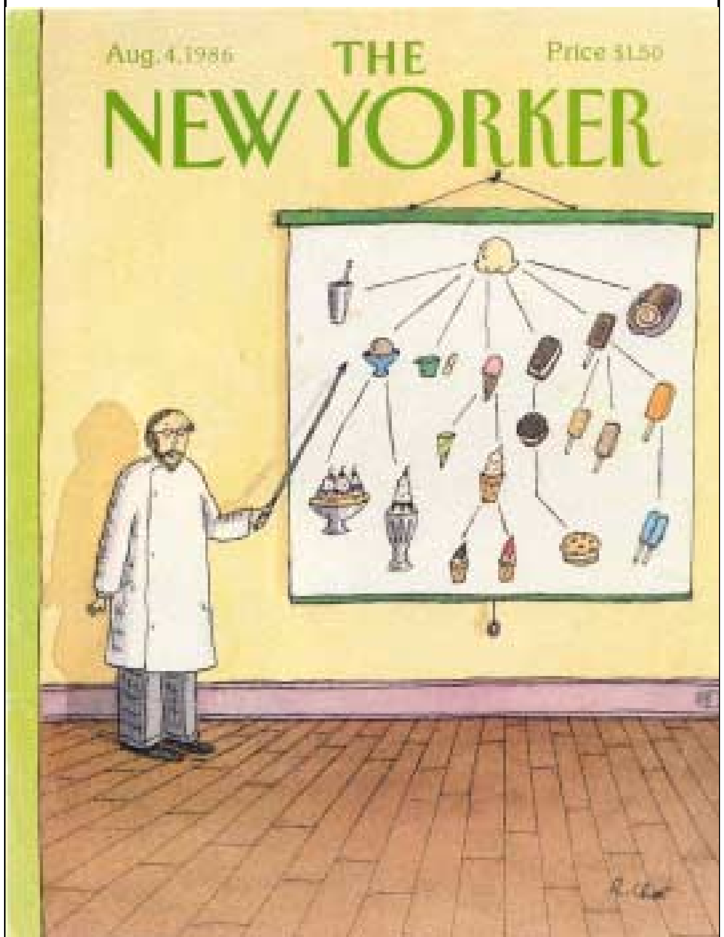
Semi-automated

Domain-independent modeling languages
- State Charts
- Interaction Diagrams
- Activity Diagrams

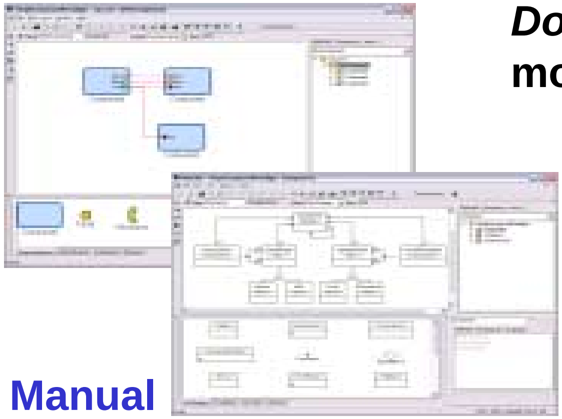
Technology Evolution (3/4)

Level of Abstraction

Programming Languages & Platforms



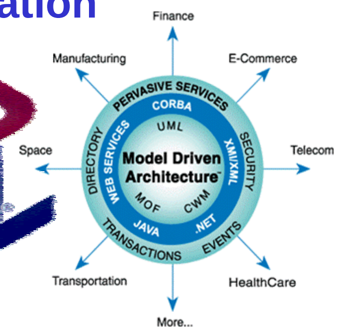
Model-Driven Engineering (MDE)



Domain-specific modeling languages

- ESML
- PICML
- Mathematica
- Excel
- *Metamodels*

Manual translation



Domain-independent modeling languages

- State Charts
- Interaction Diagrams
- Activity Diagrams

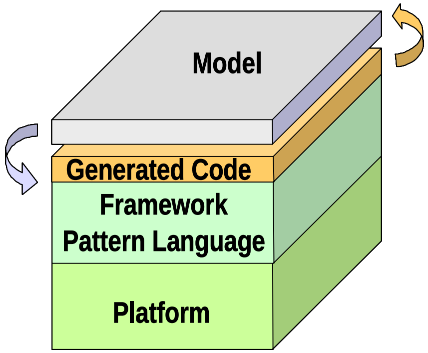
Semi-automated

- OMG is evaluating MDE via MIC PSIG
  - mic.omg.org

Technology Evolution (3/4)

Level of Abstraction

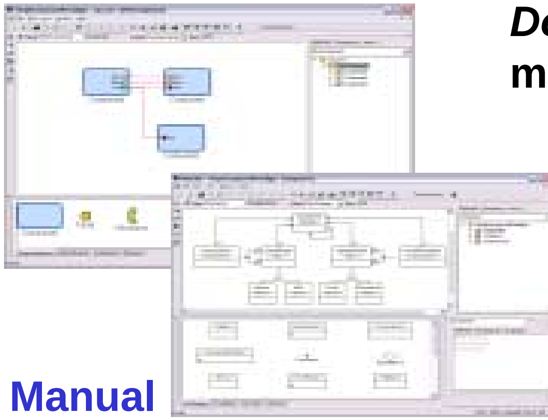
Programming Languages & Platforms



Components
Frameworks
Class Libraries
Operating Systems
Hardware

C++/Java
C/Fortran
Assembly
Machine code

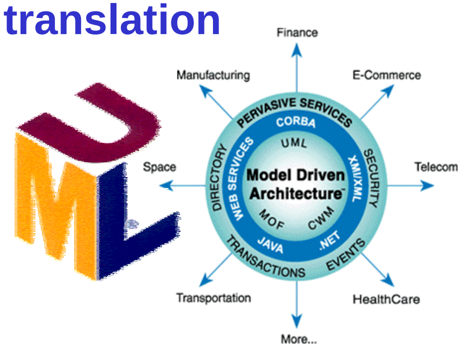
Model-Driven Engineering (MDE)



Manual translation

***Domain-specific* modeling languages**

- ESML
- PICML
- Mathematica
- Excel
- *Metamodels*



Semi-automated

Domain-independent modeling languages

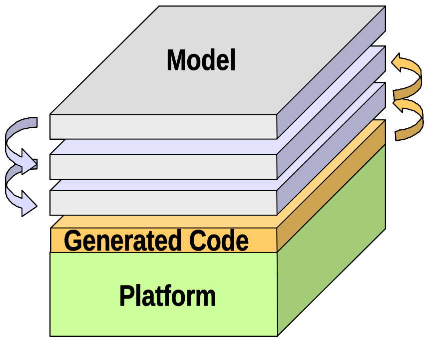
- State Charts
- Interaction Diagrams
- Activity Diagrams

- OMG is evaluating MDE via MIC PSIG
  - mic.omg.org

Technology Evolution (4/4)

Level of Abstraction

Programming Languages & Platforms

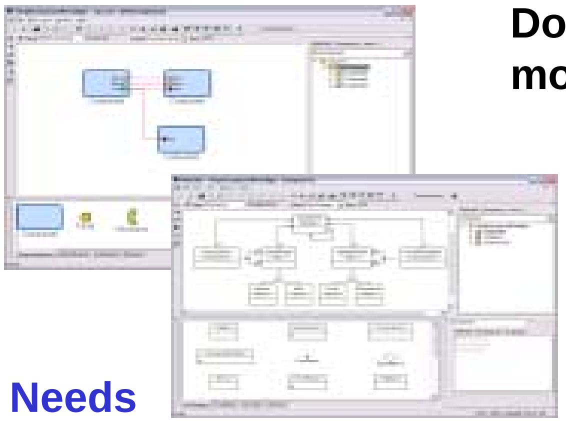


Needs Automation

Components
Frameworks
Class Libraries
Operating Systems
Hardware

C++/Java
C/Fortran
Assembly
Machine code

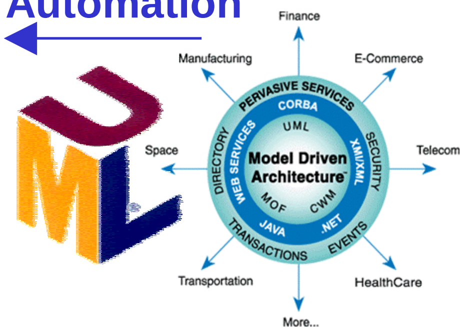
Model-Driven Engineering (MDE)



Domain-specific modeling languages

- ESML
- PICML
- Mathematica
- Excel
- *Metamodels*

Needs Automation



Domain-independent modeling languages

- State Charts
- Interaction Diagrams
- Activity Diagrams

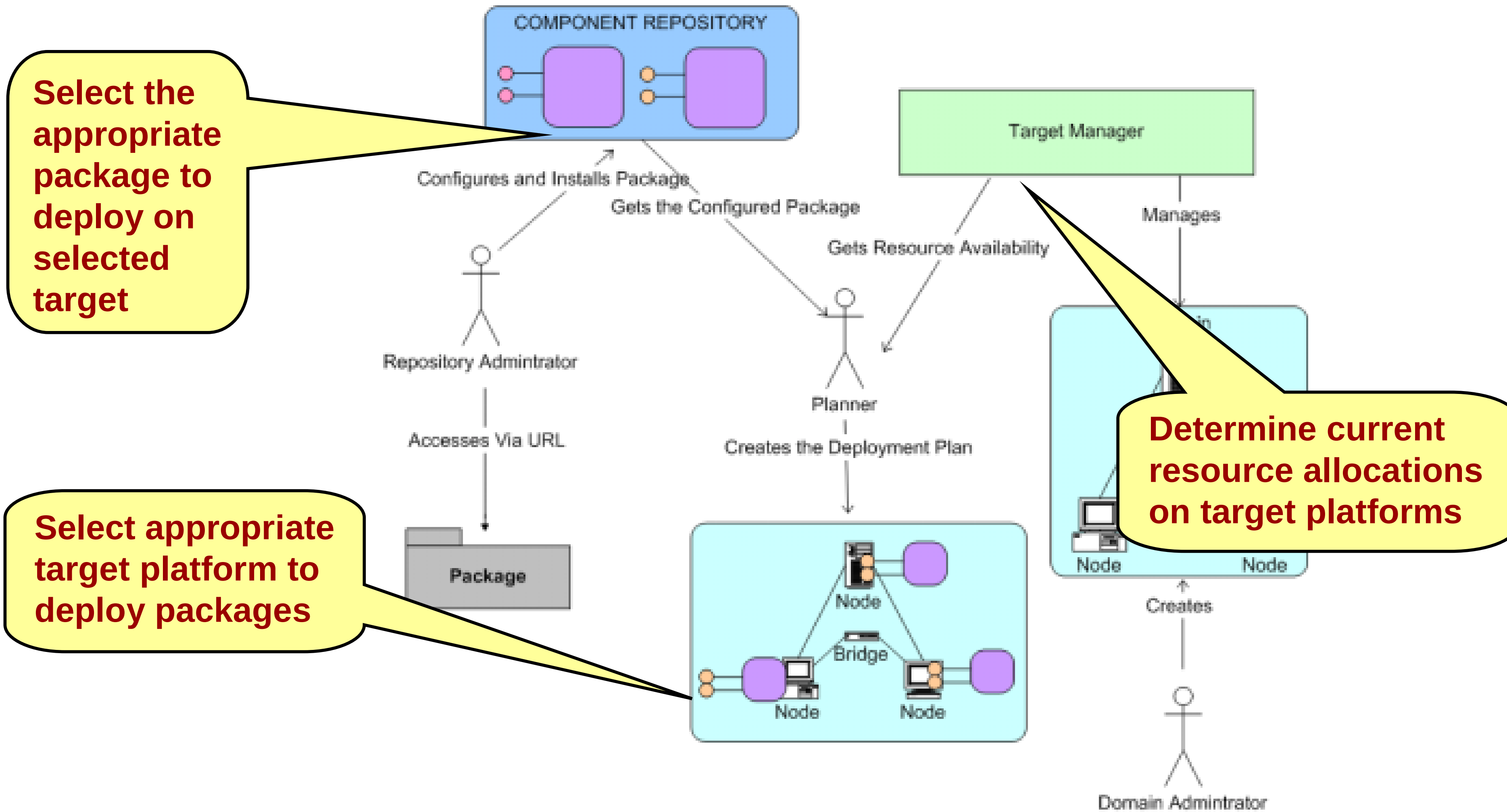
Needs Automation

Research is needed to automate DSMLs & model translators

See February 2006 IEEE Computer special issue on MDE techniques & tools

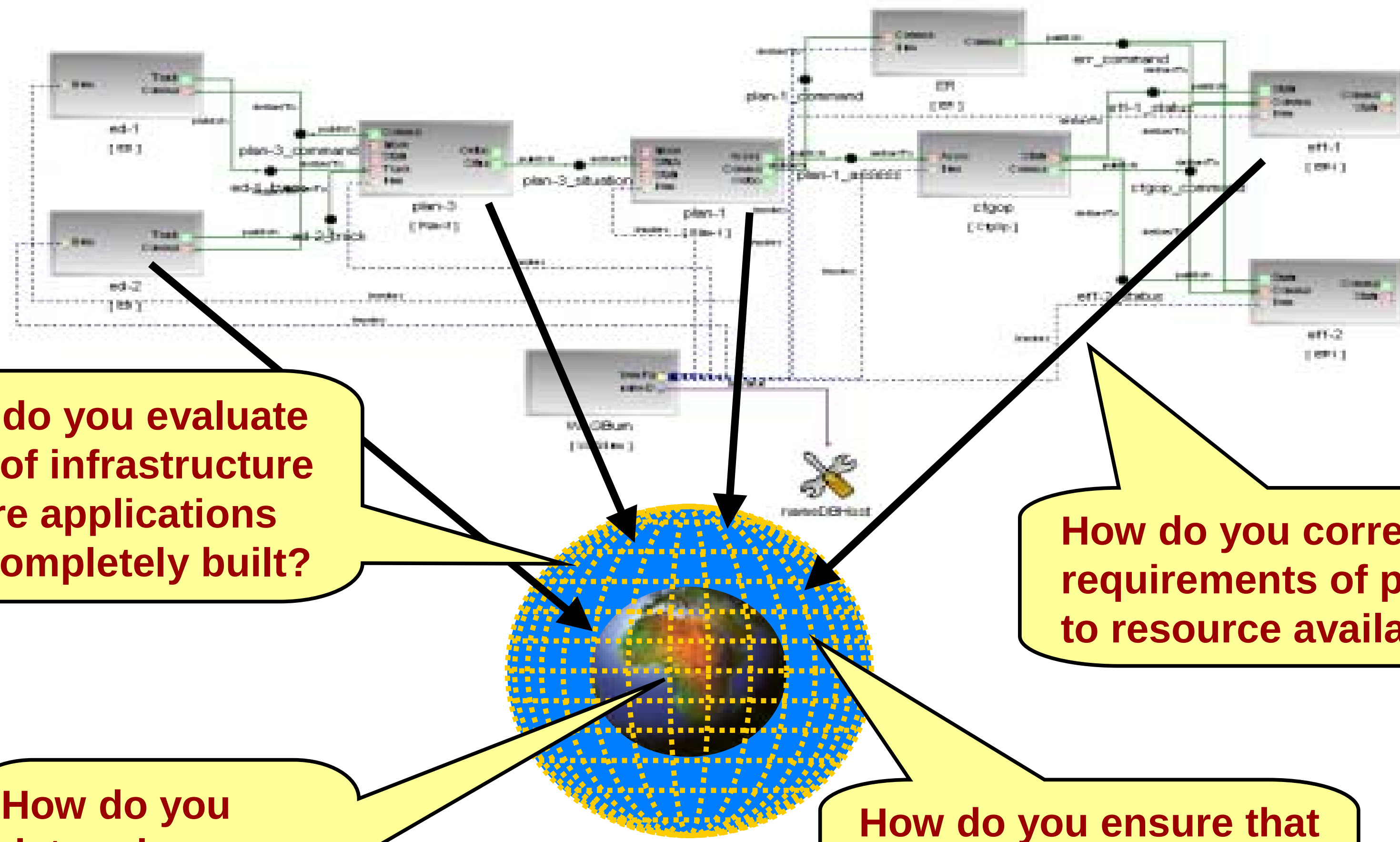
ULS Systems Challenge: Planning Aspect

System integrators must make appropriate deployment decisions, identifying nodes in target environment where packages will be deployed



Planning Aspect Problems

Ensuring deployment plans meet ULS system QoS requirements



How do you evaluate QoS of infrastructure before applications are completely built?

How do you correlate QoS requirements of packages to resource availability?

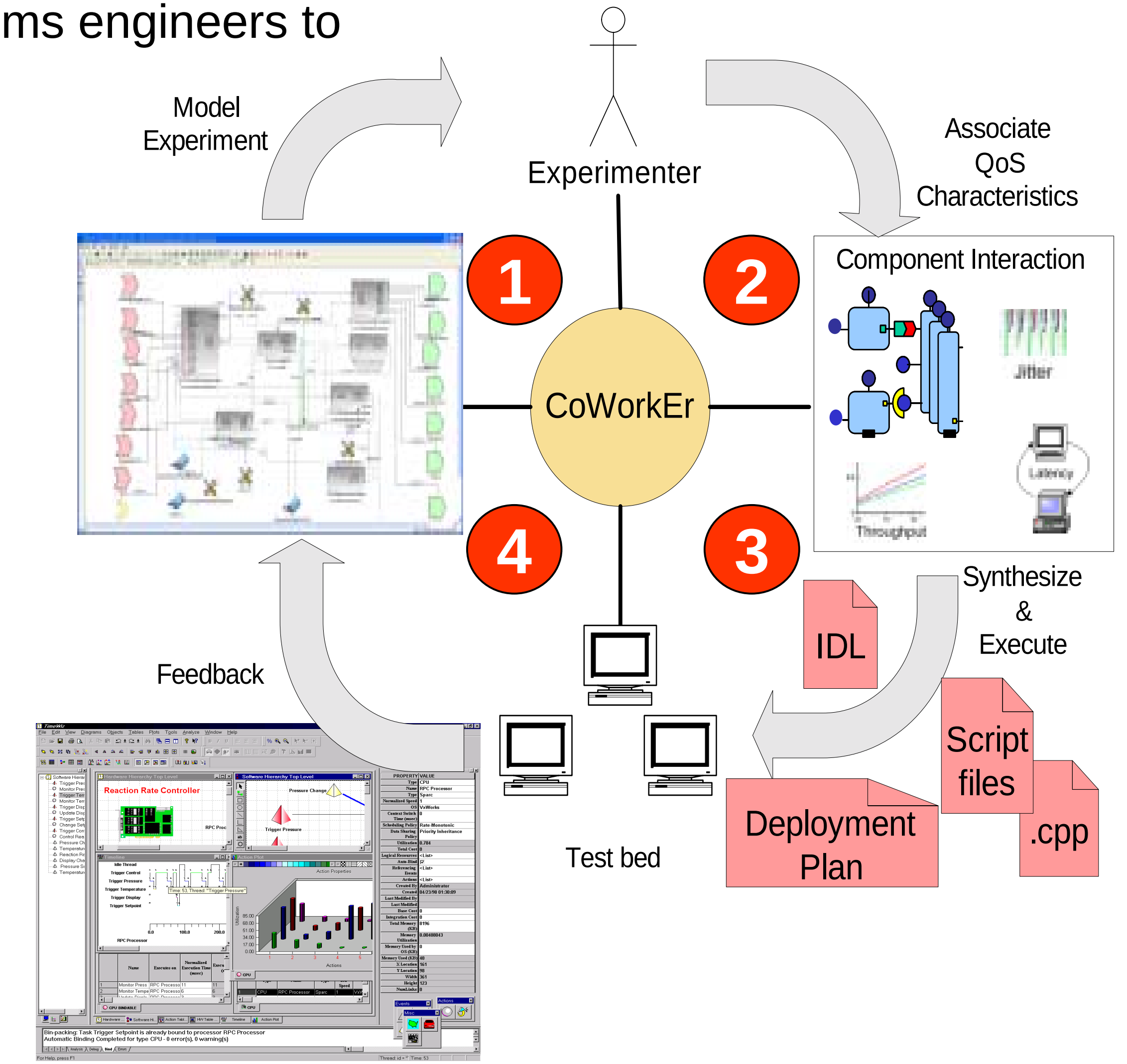
How do you determine current resource allocations?

How do you ensure that selected targets will deliver required QoS?

SEM Tool Approach for Planning Aspect

Approach

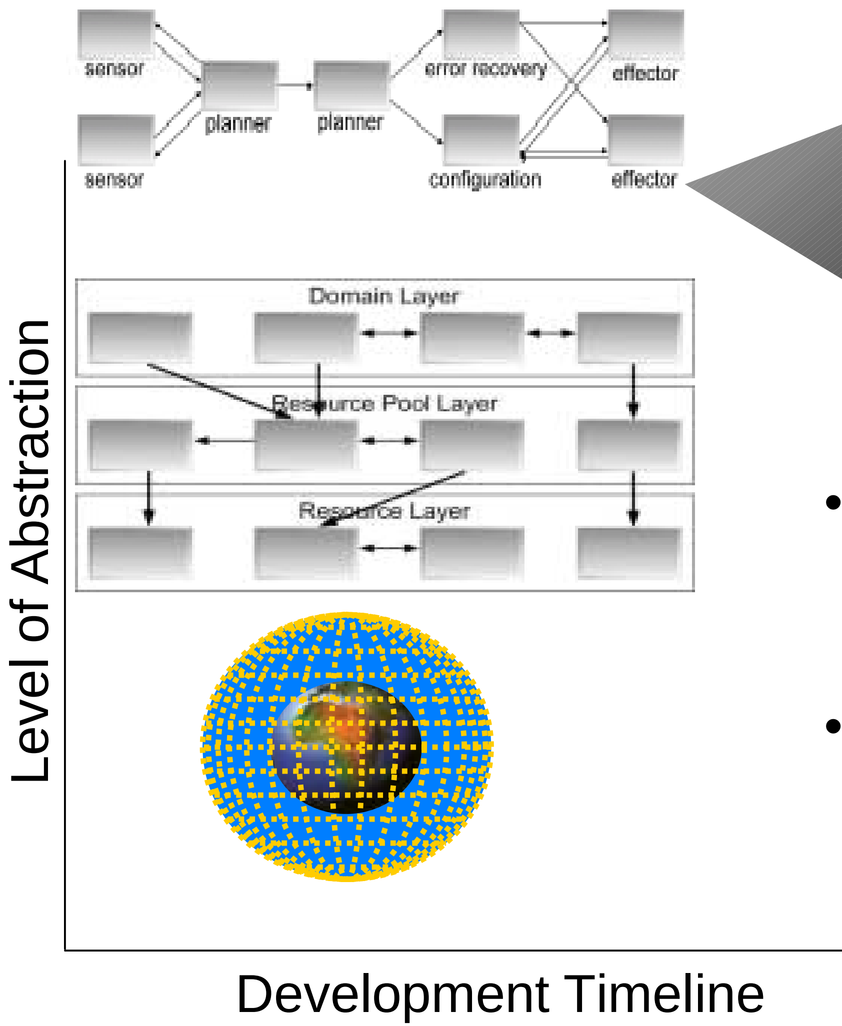
- Develop ***Component Workload Emulator (CoWorkEr) Utilization Test Suite (CUTS)*** to allow architects & systems engineers to

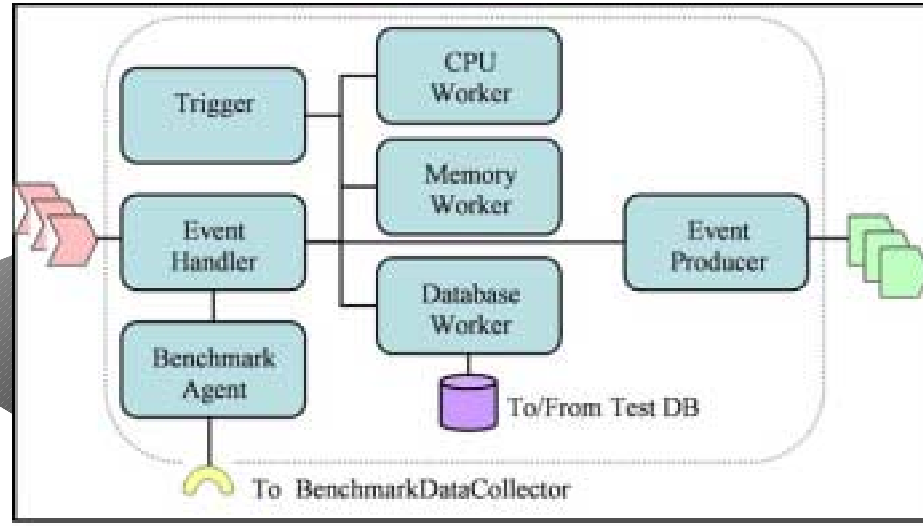


1. Compose scenarios to exercise critical system paths
2. Associate performance properties with scenarios & assign properties to components specific to paths
3. Configure workload generators to run experiments, generate deployment plans, & measure performance along critical paths
4. Analyze results to verify if deployment plan & configurations meet performance requirements

CUTS helps to conduct “what if” analysis on evolving systems

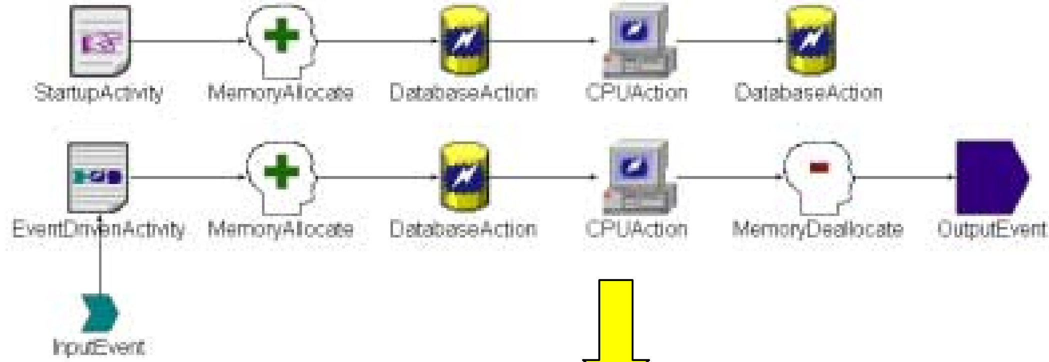
Emulating Computational Components in CUTS



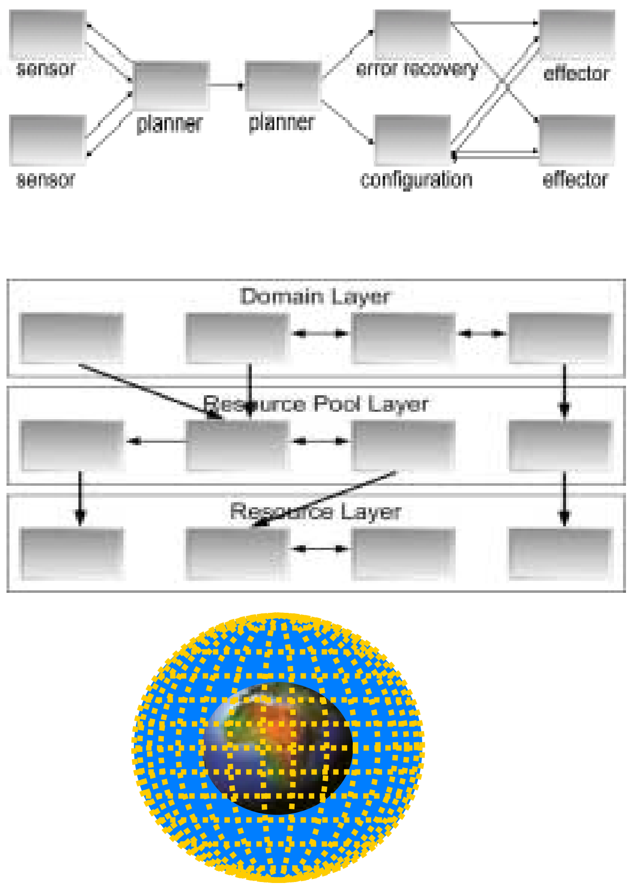


- Application components are represented as *Component Workload Emulators* (*CoWorkErs*)
- *CoWorkErs* can be interconnected by the *PICML* tool to form *operational strings*

Representing Computational Components in CUTS




```
- <EventReactionSpec>
    <InputEvent eventType="AssessmentSimEvent" count="1" />
  - <Workload>
      <MemoryAction repetitions="5" operation="ALLOCATE" />
      <DatabaseAction repetitions="40" />
      <CPUAction repetitions="14" />
      <MemoryAction repetitions="5" operation="DEALLOCATE" />
      <PublicationAction repetitions="1" eventType="CommandSimEvent" dataSize="128" />
    </Workload>
  </EventReactionSpec>
```

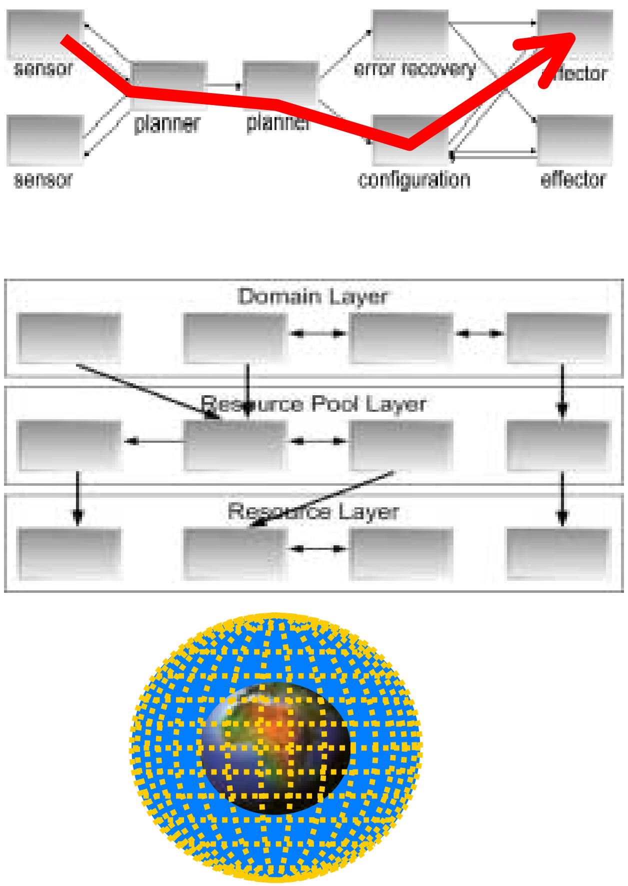



Level of Abstraction

- *Workload Modeling Language (WML)* MDE tool defines behavior of *CoWorkErs* via “work sequences”
- WML programs are translated into XML characterization files
- These files then configure *CoWorkErs*

Development Timeline

Visualizing Critical Path Performance in CUTS



Level of Abstraction

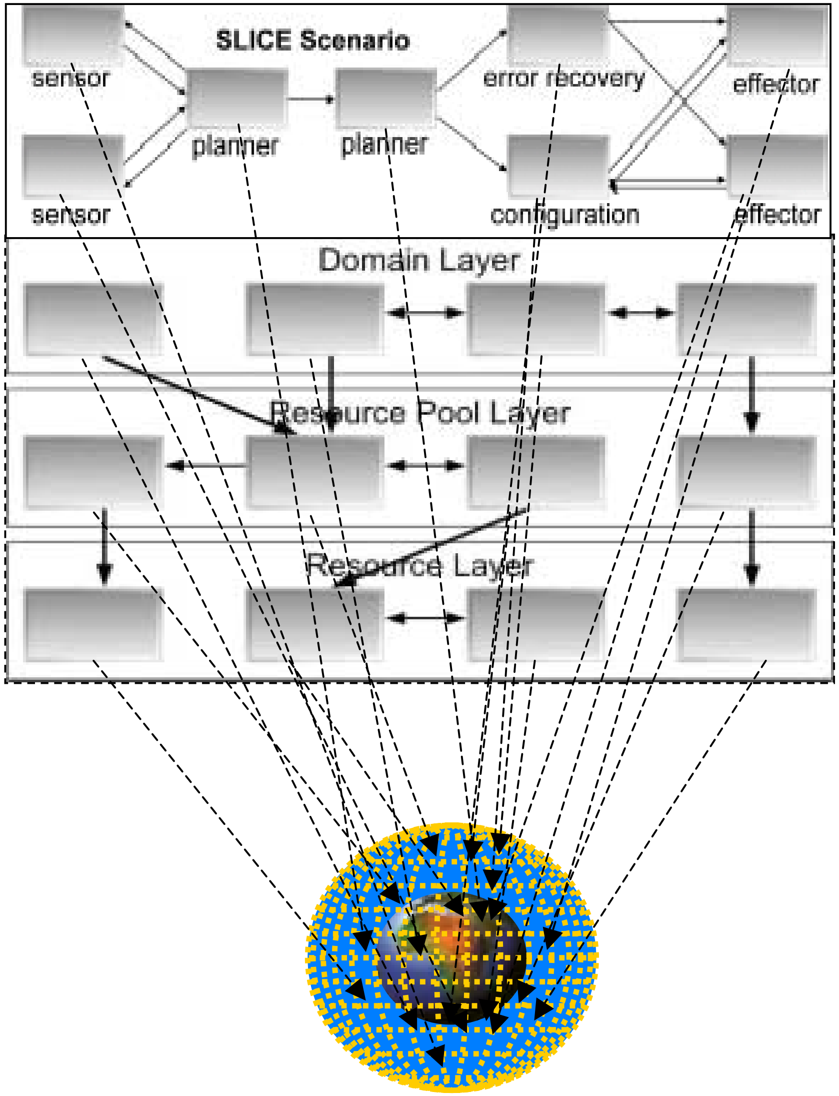
Development Timeline

- *BenchmarkManagerWeb-interface (BMW)* MDE tool generates statistics showing performance of actions in each *CoWorkEr*
- Critical paths show end-to-end performance of mission-critical operational strings

CUTS integrates nicely with *continuous integration servers*

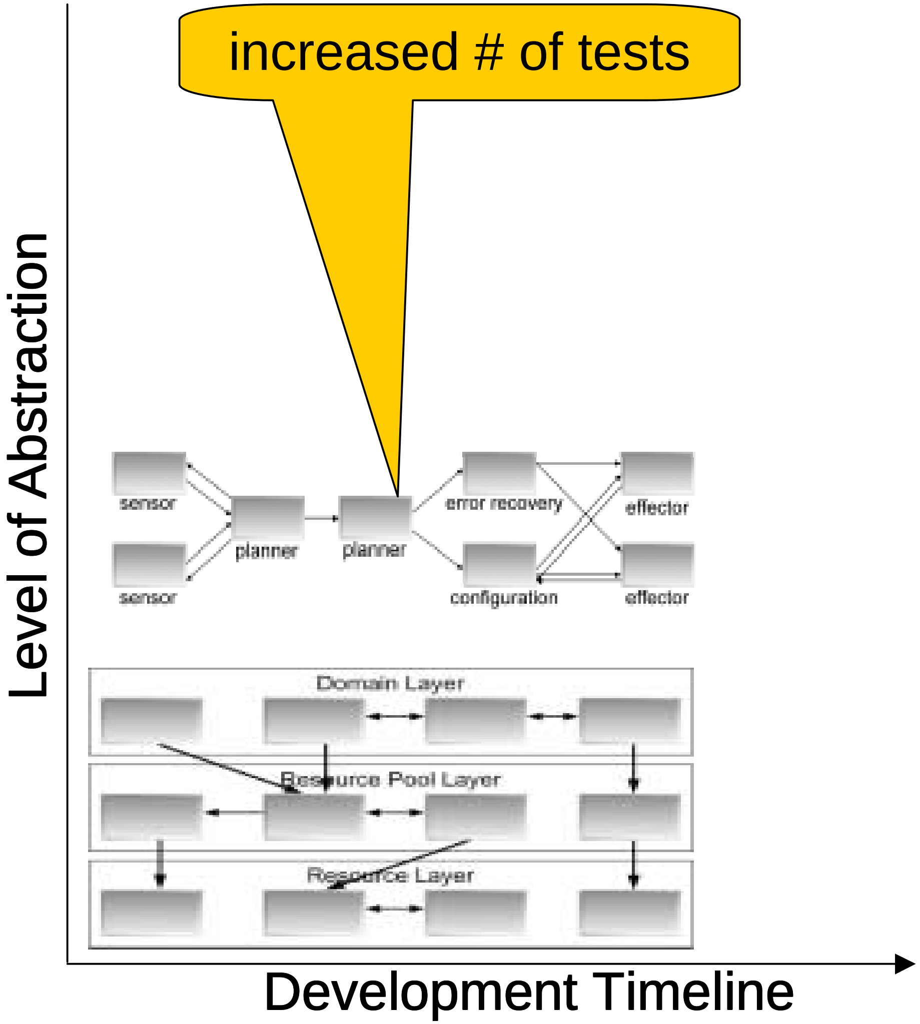
Lessons Learned Applying SEM Tools in Practice

- Component middleware technologies allowed us to leverage the behavior & functionality of target architecture for realistic emulations
- Component technologies allowed us to focus on the "business" logic of *CoWorkErs*
  - e.g., D&C handled by underlying SEM tools & middleware platforms



Lessons Learned Applying SEM Tools in Practice

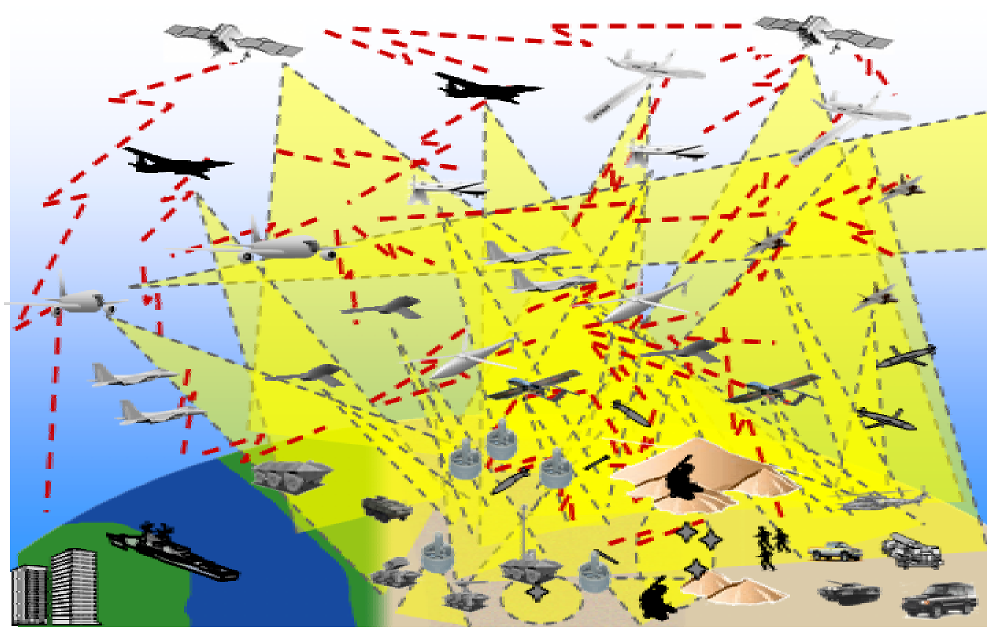
- Component middleware technologies allowed us to leverage the behavior & functionality of target architecture for realistic emulations
- Component technologies allowed us to focus on the "business" logic of *CoWorkErs*
  - e.g., D&C handled by underlying SEM tools & middleware platforms
- CUTS allowed us to test deployments *before* full system integration
- CUTS allowed us to rapidly test deployments that would have take *much* longer using *ad hoc* techniques
  - e.g., hand-coding the D&C of



CUTS is apropos when some feedback early is better than perfect feedback later

Concluding Remarks

- The emergence of ULS systems requires significant innovations & advances in tools & platforms
- Not all technologies provide the precision we're accustomed to in legacy real-time systems
- Model-driven engineering (MDE) addresses key ULS systems challenges
- Significant MDE groundwork laid in recent DARPA programs



- System execution modeling tools: www.dre.vanderbilt.edu/cosmic

More MDE info available at OMG RTW

- James Hill's CUTS demo in Demo Area
- John Slaby's talk Wed at 10am
- Kitty Balasubramanian's talk Wed. at 11:30am
- Open-source DRE middleware BoF Wed at 8pm

Much more R&D needed for ULS systems, e.g., recent Army/SEI study