Formal Approaches to Cyber Physical System Development

Jin Hyun Kim

In This Topic

- Introduction to Formal Approach (Part 1)
- Logics System for Formal Specification (Part 2)
  - Proposition Logic
  - Predicated Logic
- Uppaal – Model checking (Part 3)

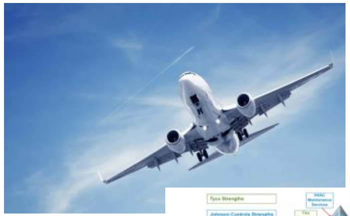
References

- Course materials from
  - CIS441/CIS541: Embedded Systems for Life-Critical IoT/CPS Applications in University of Pennsylvania

Cyber Physical Systems

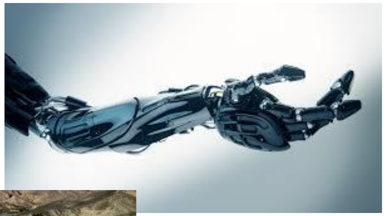
- Cyber-Physical Systems?
    - A system that controls physical components by cyber-based commands
    - It is a physical system whose operations are integrated, monitored, and/or con trolled by a computational core [E1].

Aircraft



Medical Devices

Robotics



Smart Buildings

Automobile

[E1] “Cyber physical systems,” National Science Foundation, 2014. [Online]. Available: http://www.nsf.gov/pu blications/pub summ. jsp?ods key=nsf14542

Cyber Physical Systems

Example: BMW 745i

- 2,000,000 LOC, Window CE OS
- Over 60 microprocessors: 53 8-bit, 11 32-bit, 7 16-bit
- Multiple networks

SW intensive!

National Health Information Network, Electronic Patient Record initiative

- Medical records at any point of service

Operating Room of the Future

- Closed loop monitoring and control; multiple treatment stations, plug and play devices; robotic microsurgery

Human-related!

Key Trends in CPS

- System complexity
  - Increasing functionality, integration, networking interoperability,
  - Growing importance and reliance on software (SW)
  - Increasing non-functional constraints
- System dynamicity
  - Dynamic, ever-changing, dependable, high-confidence
  - Self-*(aware, adapting, repairing, sustaining)
- Cyber-Physical Systems everywhere, used by everyone, for everything
  - 24/7 availability, 100% reliability, 100% connectivity, instantaneous response, …
- Interoperability between human and systems

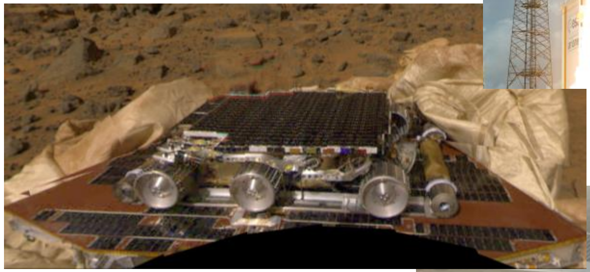
Challenges in CPS

- How can we provide people and society with cyber-physical systems that they can trust their lives on?

Complex system failures

- Denver baggage handling system ($300M)
- Power blackout in NY (2003)
- Ariane 5 (1996)
- Mars Pathfinder (1997)
- Mars Climate Orbiter ($125M,1999)
- The Patriot Missile (1991)
- USS Yorktown (1998)
- Therac-25 (1985-1988)
- London Ambulance System (£9M, 1992)
- Pacemakers (500K recalls during 1990-2000)
- Numerous computer-related incidents with commercial aircraft (https://www.fss.aero/accident-reports/browse_type.php?type=operator)






New Challenges

Deep neural networks (DNNs) to directly instruct physical effectors of cyber-physical systems

PilotNet (NVIDIA, 2016)

ACAS Xu DNN (Stanford, 2016)

ChauffeurNet (Google, 2018)

ACAS sXu (NUAIR, 2018)

ANYmal (ETH Zurich, 2019)

Handle (Boston Dynamics, 2019)

New Challenges: Unsafe AI


89 deaths and 57 injuries

Toyota ETCS bugs (2009~2011)

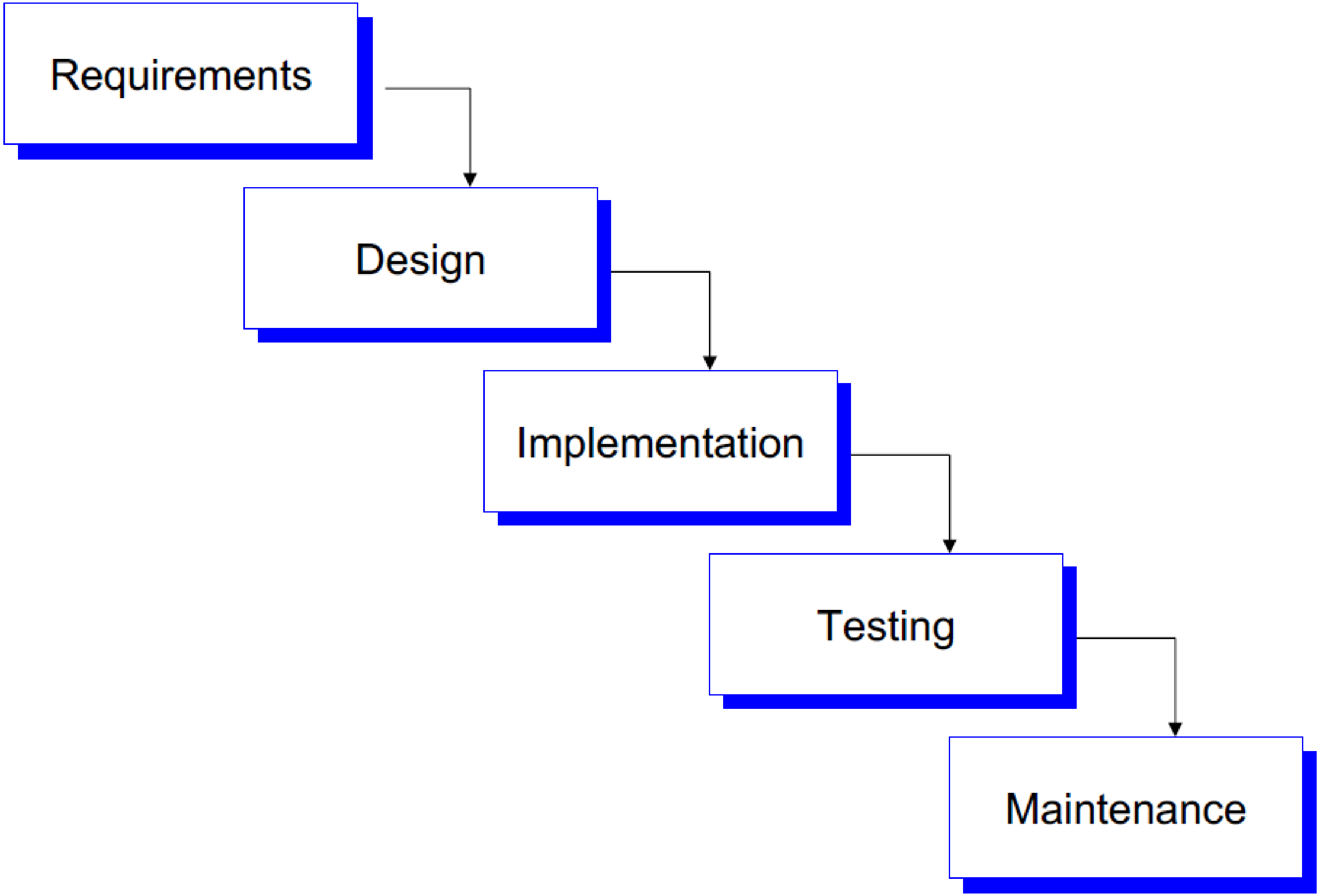

4 deaths (drivers)

Tesla Autopilot (2016~2019)

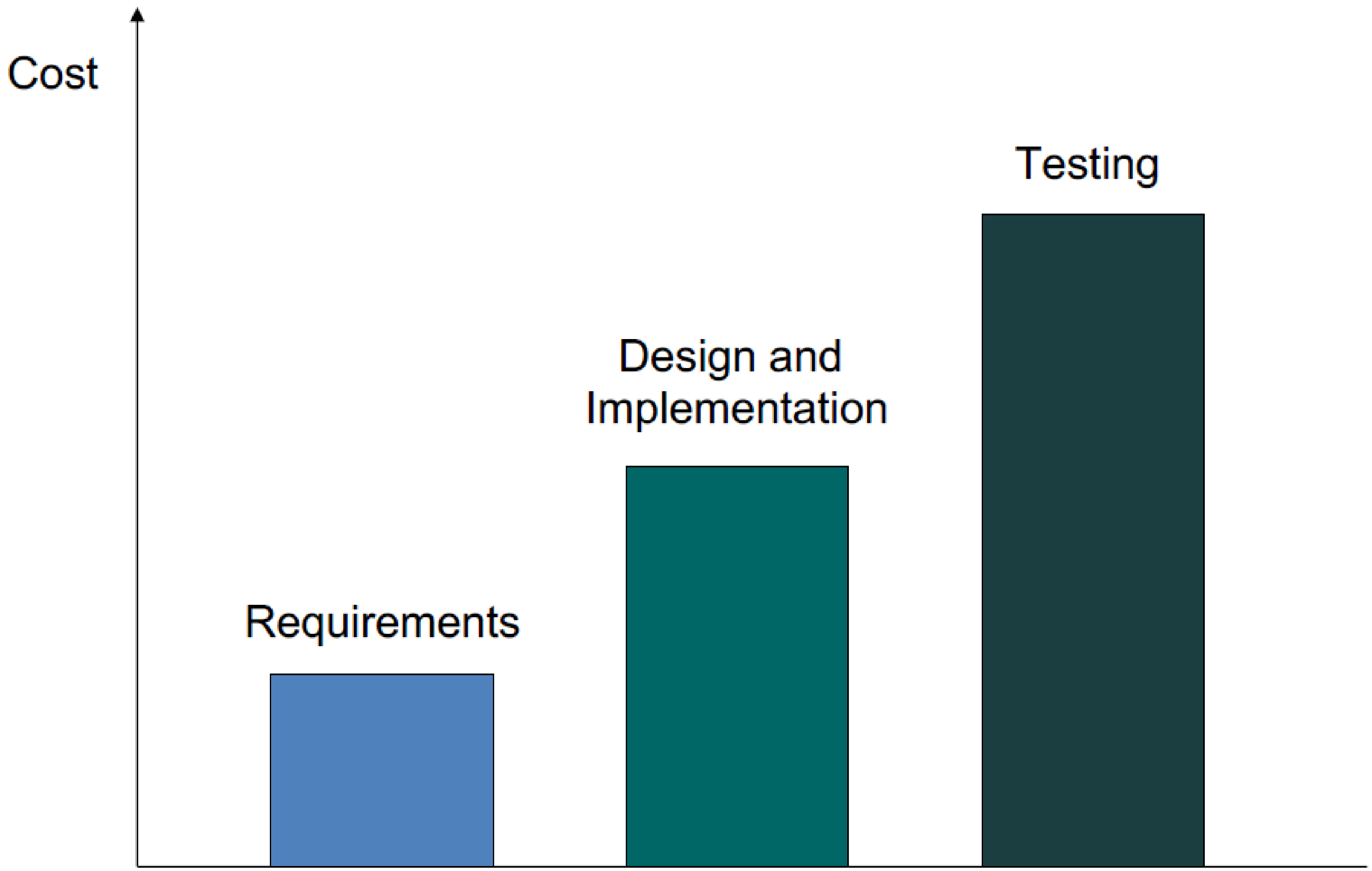

1 death (pedestrian)

Uber Self-Driving Testing (2018)

Software Life Cycle

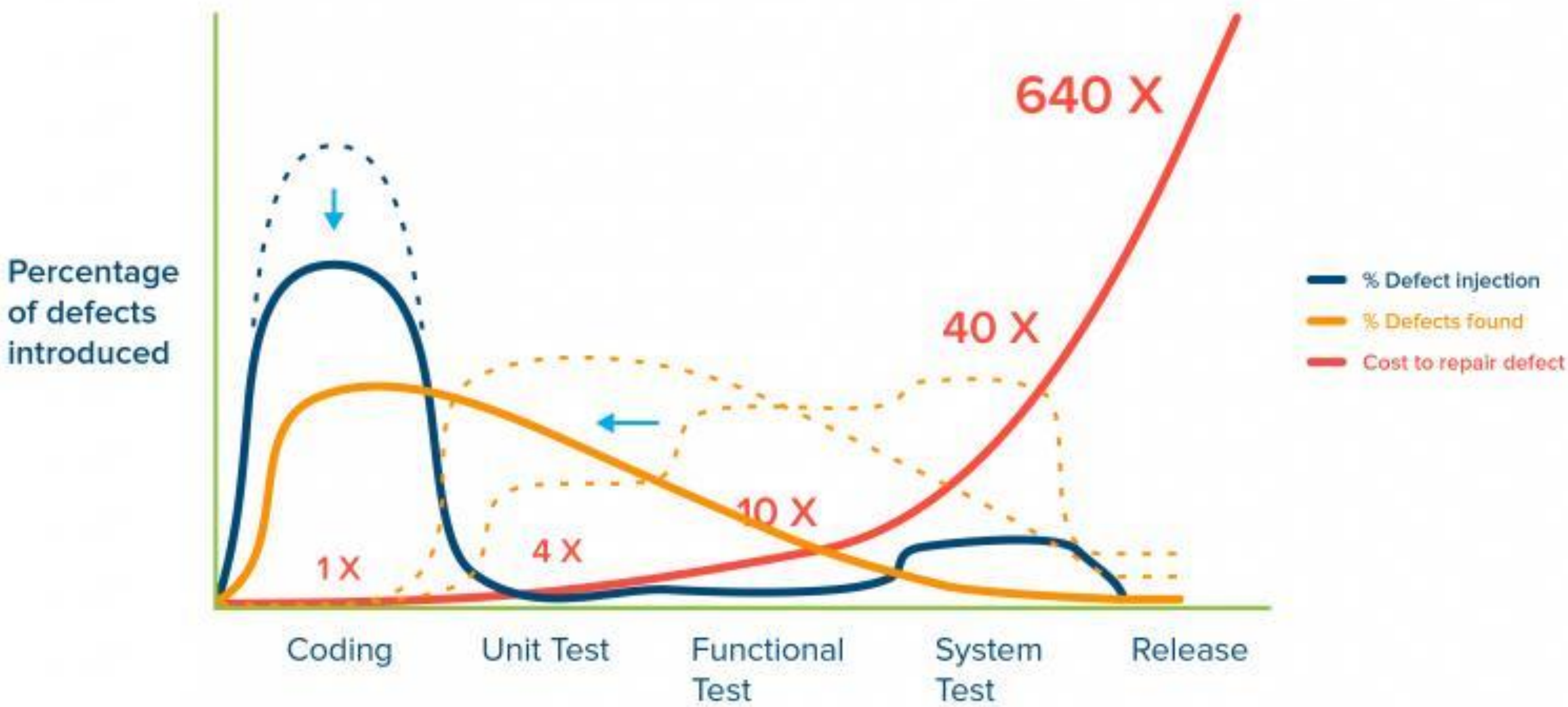


Software Development Costs



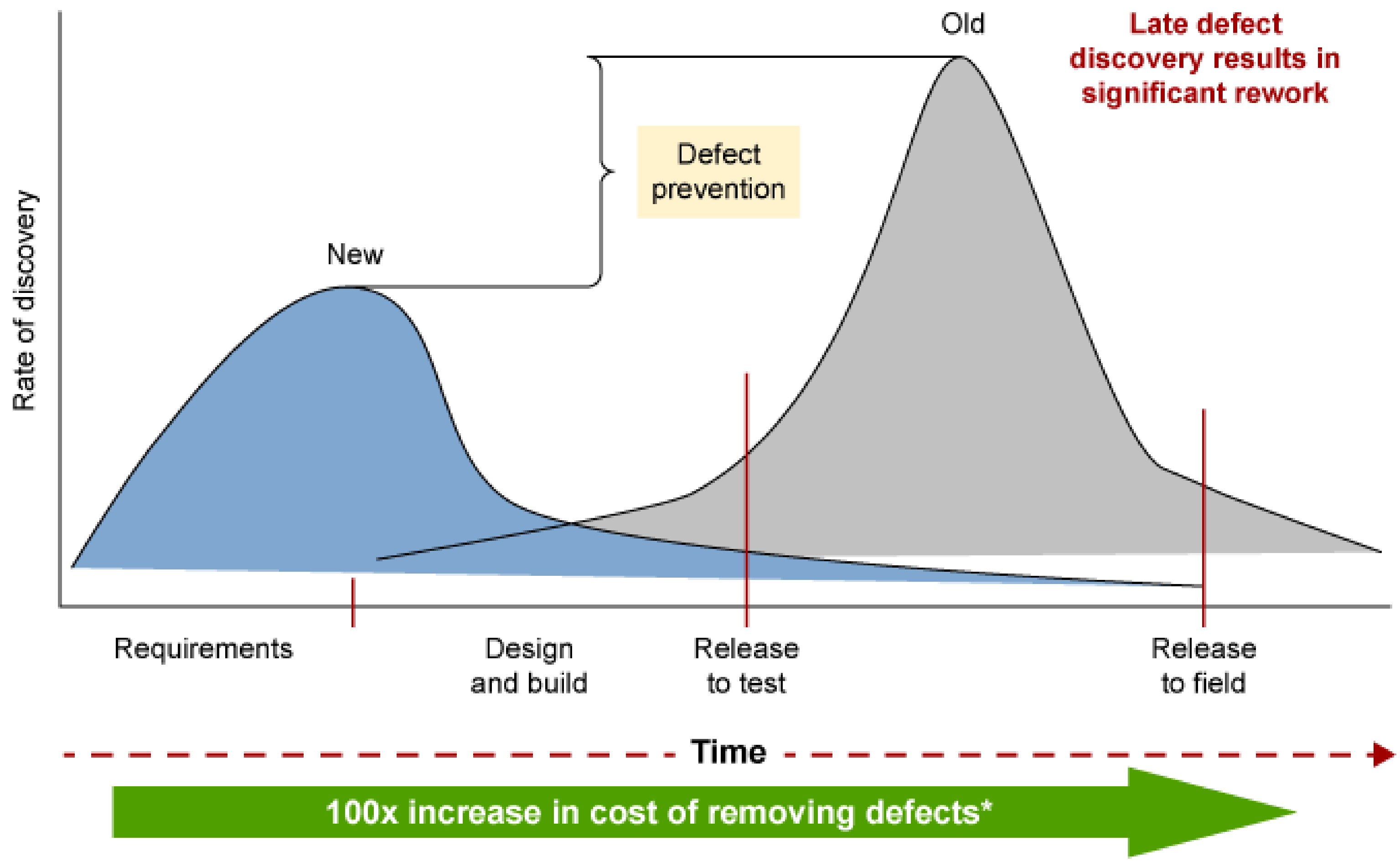
결함 비용

https://www.stickyminds.com/article/shift-left-approach-software-testing



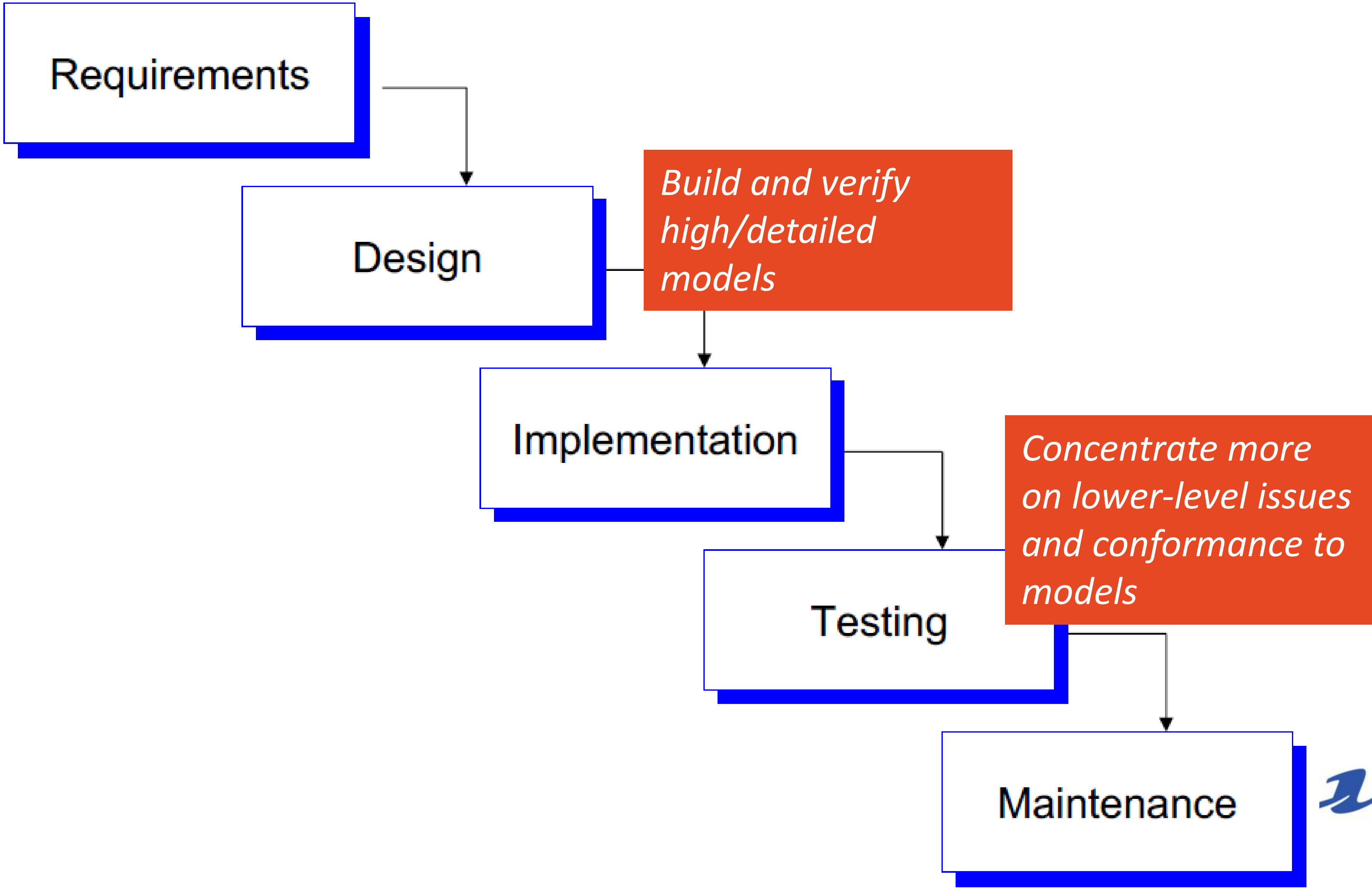
Jones, Capers. *Applied Software Measurement: Global Analysis of Productivity and Quality.*

결함비용

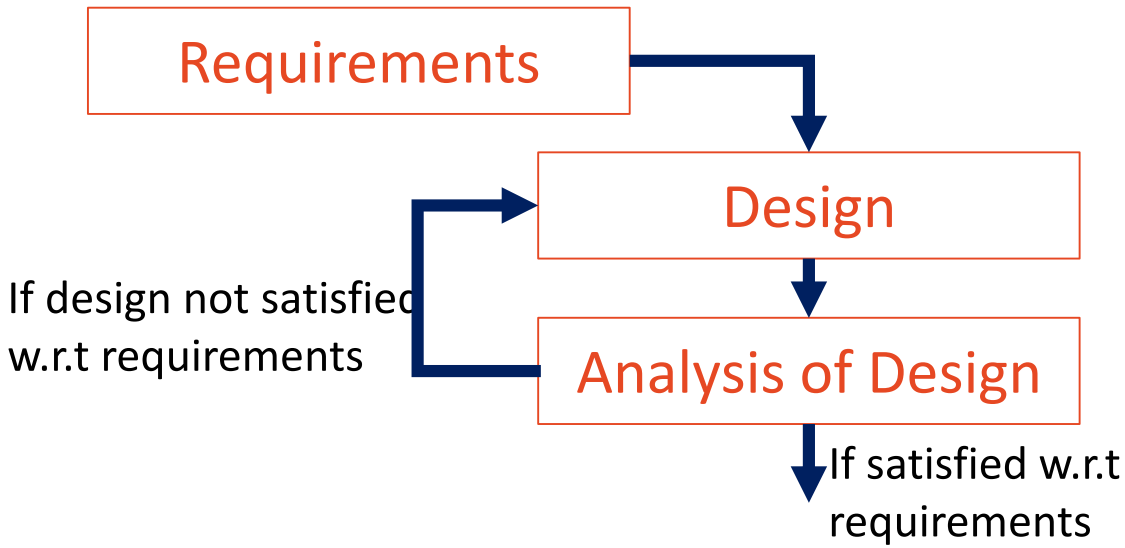


*Source: Boehm, Barry. *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice-Hall, Inc. 1981.
Boehm, Basili, "Software Management." *IEEE Computer*, January 2001.

Model-based Development



Design Process



- Requirements specify **what** a system is supposedto do
- Design describes **how** it does it
- Analysis says **why** “how” meets “what” (i.e.,design satisfies requirements)

Software Non-Functional Requirements

- Correctness
- Reliability (dependability)
- Robustness
- Safety
- Security
- Performance
- Productivity
- Maintainability, portability, interoperability, …

Software Verification and Validation

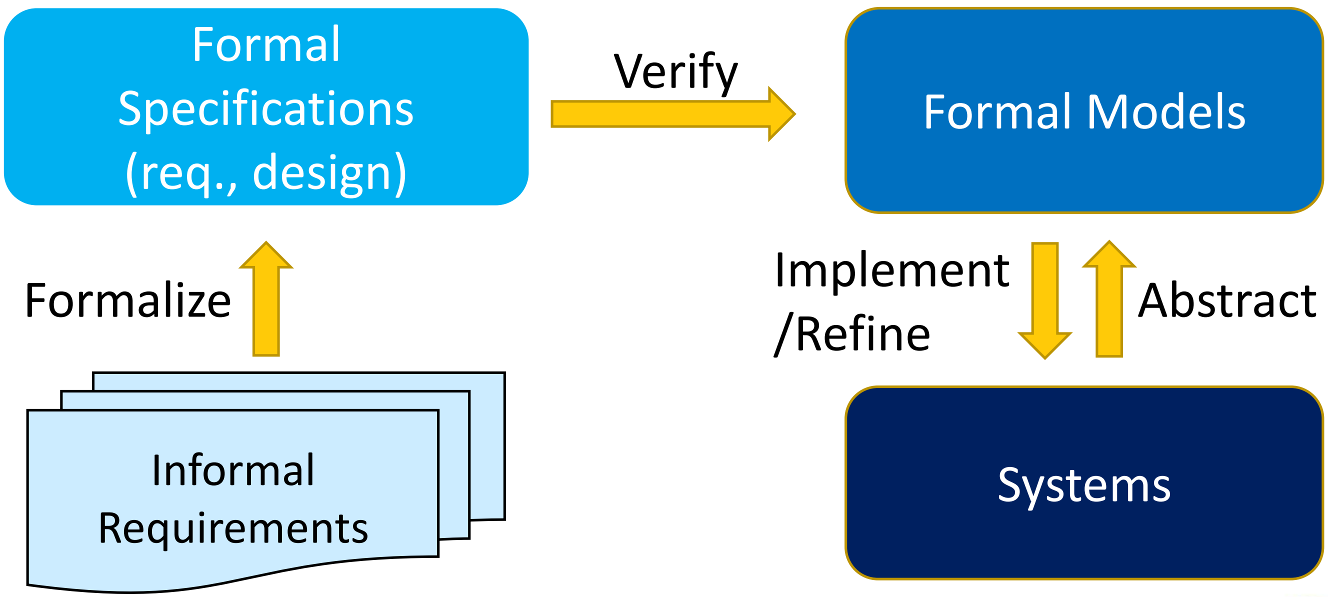
- Verification
  - Are we building the product right?
  - Process-oriented
  - Does the product of a given phase fulfill the requirements established during the previous phase?
- Validation
  - Are we building the right product?
  - Product-oriented
  - Does the product of a given phase fulfill the user's requirements?

Techniques for V&V

- Static
  - Collects information about software without executing it
  - Reviews, walkthroughs, and inspections
  - Static analysis
  - Formal verification
- Dynamic
  - Collects information about software with executing it
  - Testing: finding errors
  - Debugging: removing errors
  - Runtime verification

Model-Based Formal Analysis

- From requirements to formal specification
  - Formalize specification, derive model
  - Formally verify correctness



Formal Methods

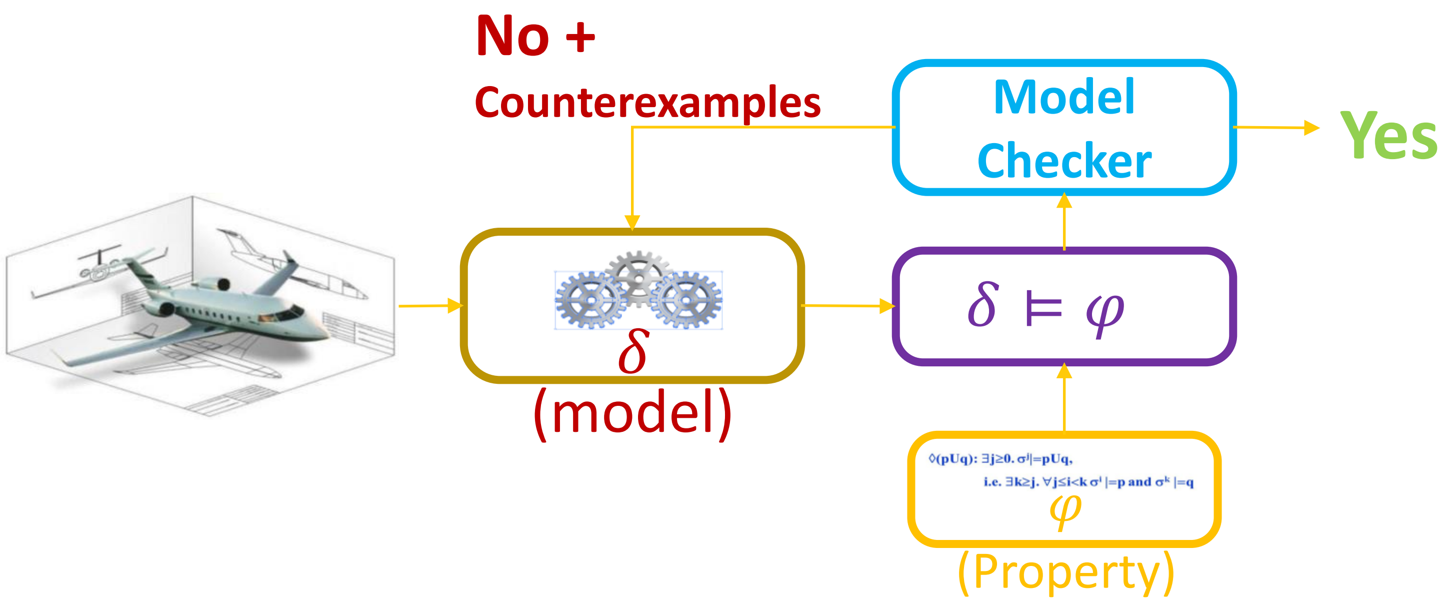
- Software engineering based on mathematical proof techniq ues
  - Check whether a property of a computational system holds for all possible executions
  - Automated model checking, theorem proving, static analysis, Run-time verification etc.
- Compared to testing techniques,
  - ***Testing*** just sample a space of behaviors, but ***FM*** proves behaviors
  - 5*5-3*3 = (5-3)*(5+3) vs X^2 − y^2 = (x − y)(x + y)
- Safety-related standards and regulations, such as ISO 26262 (automotive), DO 178-B (avionics), IEC 62304 (medical device s), recommends formal methods for safety and security assur ance analysis techniques

Formal Methods

- Formal Methods
  - application of rigorous, mathematics-based techniques to establish the correctness of (computerized) systems
  - many techniques: manual proof, automated theorem proving, static analysis, ***model checking***, …
- "Testing can only show the presence of errors, not their absence." -- Edsger Dijkstra
- To rule out errors, testers should consider all possible executions
  - Need to automate the testing process?
  - Need a different method, namely formal methods!

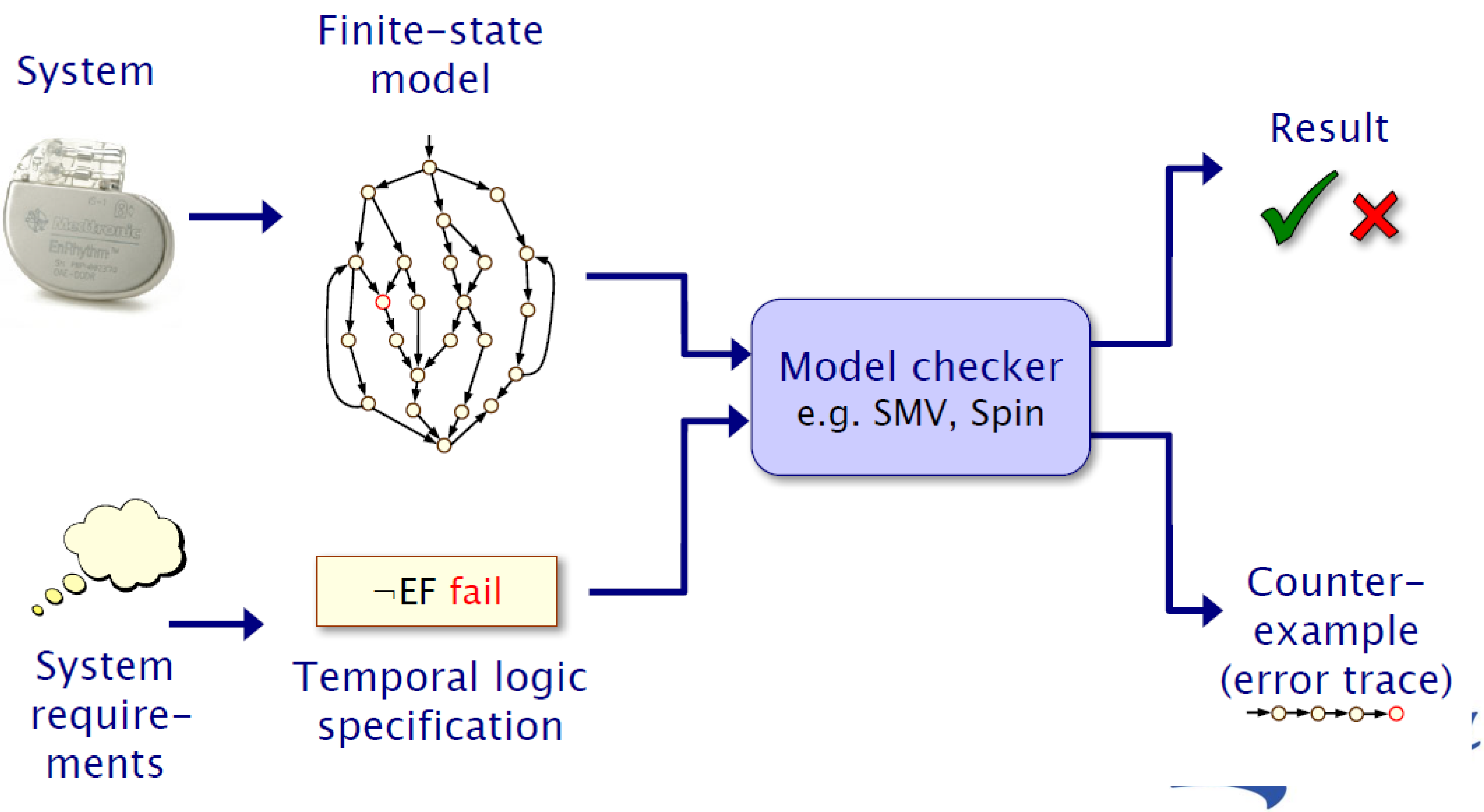
gnu

Model Checking



- A system is given as a model (σ), and a property (φ) is also specified, then model checker (MC) explores every states until any state violates the property
- If MC finds any property-violating state, it alarms with an counterexample to trace the state

Model Checking



Merit of Model Checking

- Model Checking simple properties (e.g., deadlock freeness) is already extremely useful.
- The goal is no longer seen as proving that a system is completely, absolutely, and undoutedly correct (bug-free).
  - The objective is to have tools that can help a developer find errors and gain confidence in her/his design (which is now is achievable).
- In recent years, it widely used in hardware design, protocol design, and increasingly, embedded systems!

gnu

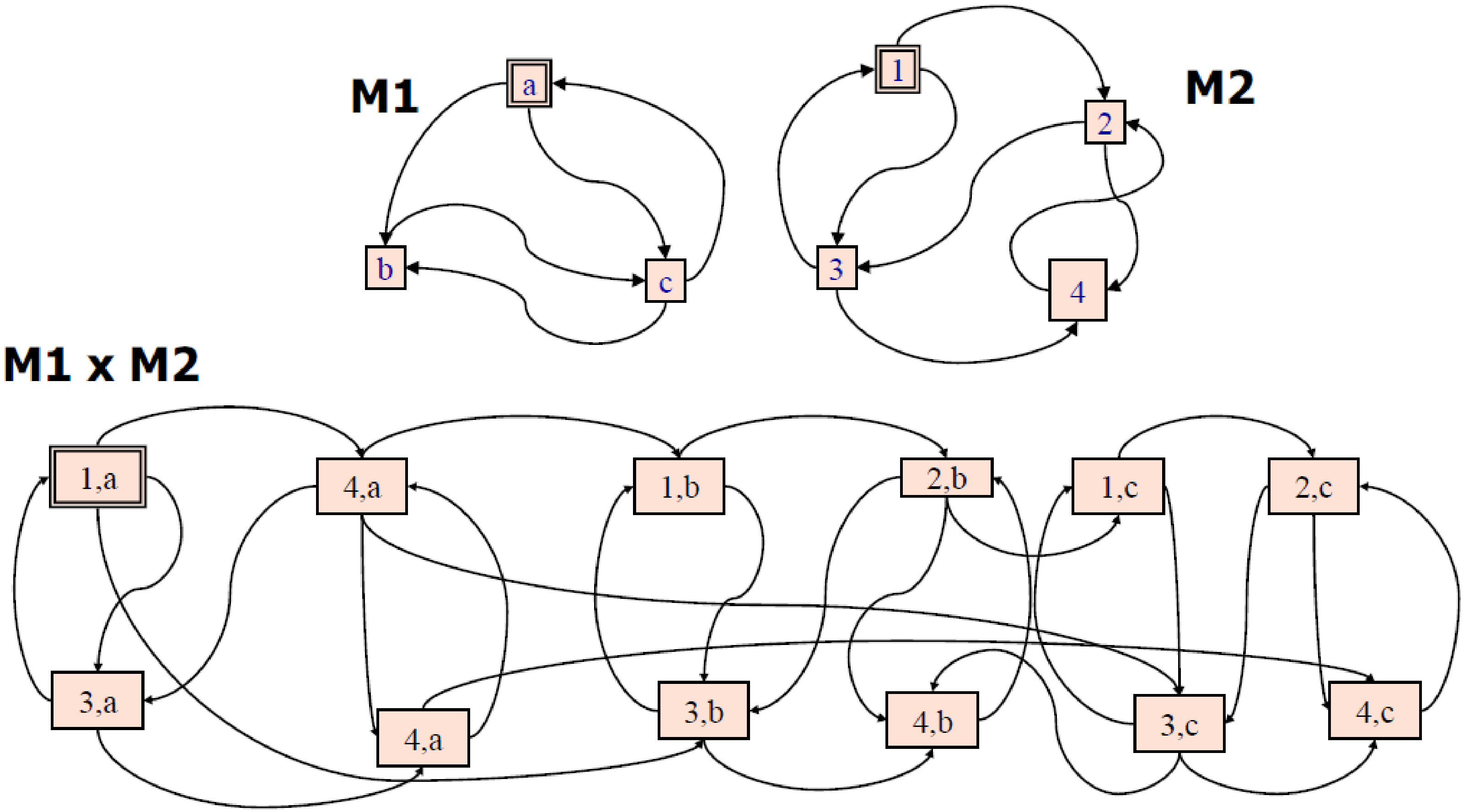
Testing/Simulation vs Model Checking

- Testing/Simulation:
  - coverage problems,
  - difficult to deal with non-determinism and concurrent computation
- Model Checking
  - exhaustive analysis of software and hardware design
  - provides 100% coverage
- Model checking may complement testing to find (design) bugs as early as possible!

Systems Verifiable by MC

- Checking correctness of
  - Communication protocols
  - Distributed Algorithms
  - Controllers
  - Hardware circuits
  - Parallel and distributed software
  - Embedded and real-time systems and software e.g., Absence of race conditions, proper synchronization, ….
- Model checking is the appropriate technique when there are many many different scenarios of interaction between concurrent components in a system

State Explosion Problem



All combinations = exponential in number of components

State Explosion Problem

- 10 components and each with 10 states with 1 clock
  - number of states = 10,000,000,000 =10 G
  - If each local state needs 4 bytes to store, then each global state needs (10 * 10)* 4Bytes = 400 Bytes

Worst case memory usage >> 4,000GB

Summary

- Conventional software lifecycle
  - 30-50% of development time/money for testing
  - Errors detected: the later the more expensive
- Model-based design & development
  - can help software developers find errors in the early stage of development lifecycle, and gain confidence in the design
- Formal Verification & Model Checking
  - the application of rigorous, mathematics-based techniques to establish the correctness of computerized systems
  - explore all possible system executions
  - increasingly used in embedded system design