Introduction

Überblick

1 Introduction	17/18/19	11.	14:00	Vortmann
2 Time and Order	24/25	11.	14:00	Vortmann
3 RT Model	**4**	**12.**	**15:00**	Vortmann
4 RT Commun	9/10	12.	14:00	Vortmann
5 FT Fundamentals	**11**	**12.**	**14:00**	**Treitl**
6 RT Operating Systems	15/16/17	12.	14:00	Vortmann
7 System Design				
8 Twelve Principles				

Outline

- When is a Computer System “Real-Time”
- Classification of Real-Time Systems
- Temporal Requirements
- What is a “System Architecture”?
- Silicon Technology Trends
- Architecture Based Design

When is a Computer System ‘Real-Time’?

A *real-time computer system* is a computer system in which the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time, when these results are produced.

The point in time when a result has to be produced is called a *deadline*.

Deadlines are dictated by the environment of the real-time computer system.

Some Definitions

If the result has utility even after the deadline, we call the deadline *soft.* Systems with soft deadlines are not the focus of these lectures.

If the result has no utility after the deadline has passed, the deadline is called *firm*.

If a catastrophe could result if a strict deadline is missed, the deadline is called *hard*.

A real-time computer system that has to meet at least one hard deadline is called a *hard real-time system.*

Hard- and soft real-time system design are fundamentally different.

Classification of RT Systems

On the basis of the external requirements

- ♦ Hard Real-Time versus Soft Real Time
- ♦ Fail-Safe versus Fail-operational

On the basis of the implementation

- ♦ Guaranteed Timeliness versus Best Effort
- ♦ Resource Adequacy-- yes or no?
- ♦ Event Triggered versus Time Triggered

Hard Real Time versus Soft Real Time

Characteristic	Hard Real Time	Soft Real Time
Response time	hard	soft
Pacing	environment	computer
Peak-Load Perform.	predictable	degraded
Error Detection	system	user
Safety	critical	non-critical
Redundancy	active	standby
Time Granularity	millisecond	second
Data Files	small/medium	large
Data Integrity	short term	long term

Fail-Safe versus Fail-Operational

A system is *fail-safe* if there is a safe state in the environment that can be reached in case of a system failure, e.g., ABS, train signaling system.
In a fail-safe application the computer has to have a high *error detection coverage.*
Fail safeness is a characteristic of the application, not the computer system.

A system is *fail operational*, if no safe state can be reached in case of a system failure,e.g., a flight control system aboard an airplane.
In fail-operational applications the computer system has to provide a minimum level of service, even after the occurrence of a fault.

Guaranteed Timeliness versus Best Effort

A system implementation provides *guaranteed timeliness* if, within the specified load- and fault-hypothesis, the temporal correctness can be substantiated by analytical arguments.

A system implementation is *best effort*, if such an analytical argument for the temporal correctness cannot be made.

The temporal verification of best effort systems relies on probabilistic arguments, even within the specified load- and fault hypothesis.

Hard real-time systems should be based on guaranteed timeliness.

Resource Adequacy

If a system has to provide guaranteed timeliness, there must be sufficient computational resources to handle the specified peak load and fault scenario.

In the past, there have been many applications where resource adequacy has been considered *too expensive*. The decreasing cost of hardware makes the implementation of resource adequate designs economically viable. In hard real-time applications, *there is no alternative to resource adequate designs*.

Predictability in Rare Event Situations

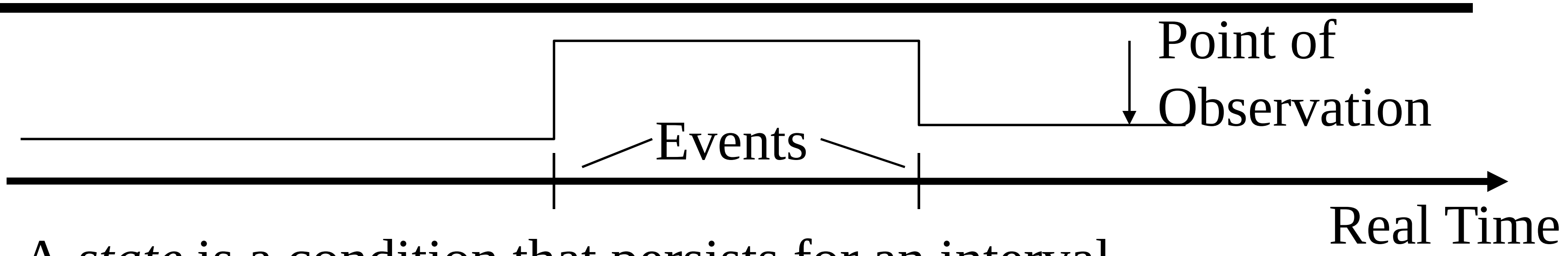
A *rare event* is an important event that occurs very infrequently during the lifetime of a system, e.g., the rupture of a pipe in a nuclear reactor.

A rare event can give rise to many correlated service requests (e.g., an alarm shower).

In a number of applications, the utility of a system depends on the predictable performance in rare event scenarios, e.g. flight control system

In most cases, workload testing will not cover the rare event scenario.

State versus Event



A *state* is a condition that persists for an interval of real time, i.e., along a section of the timeline

An *event* is an occurrence at an instant.

State information informs about the attributes of states at the point of observation (itself an event).

Event information informs about the difference in the attributes of the states immediately before and after the occurrence of the event and an estimation of the point in time of event occurrence

Only the consequences of an event can be observed.

Time Triggered (TT) vs. Event Triggered (ET)

A Real-Time system is *Time Triggered* (TT) if the control signals, such as

- ♦ sending and receiving of messages
- ♦ recognition of an external state change

are derived solely from the progression of a (global) notion of time.

A Real-Time system is *Event Triggered* (ET) if the control signals are derived solely from the occurrence of events, e.g.,

- ♦ termination of a task
- ♦ reception of a message
- ♦ an external interrupt

Temporal Requirements

Temporal accuracy of real-time data: the data elements that are displayed to the operator must be *temporally accurate*.

Maximum response time: The maximum real-time interval between a stimulus and the response must be known and bounded.

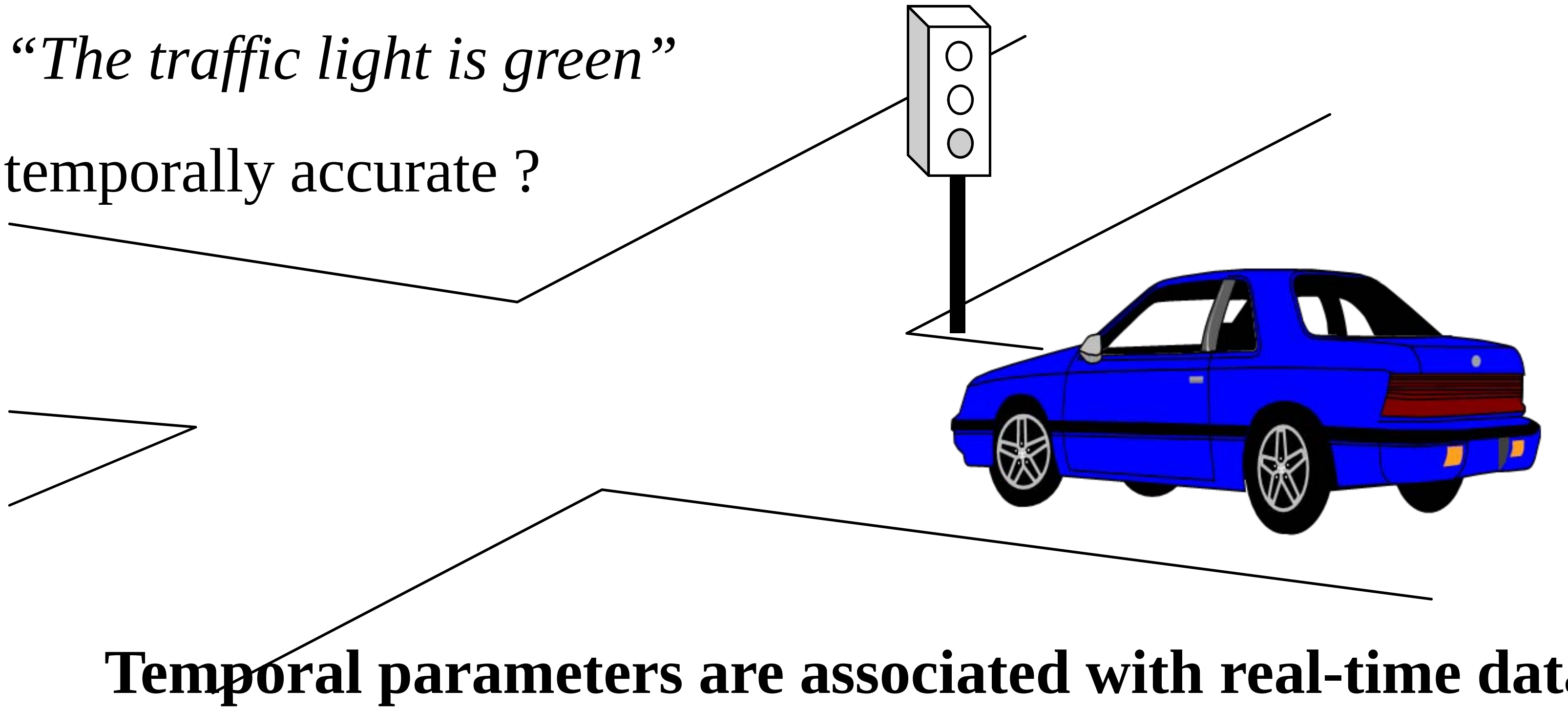
Predictability: The temporal behavior must be predictable, even in a rare event scenario.

Validity of Real-Time Information

How long is the observation:

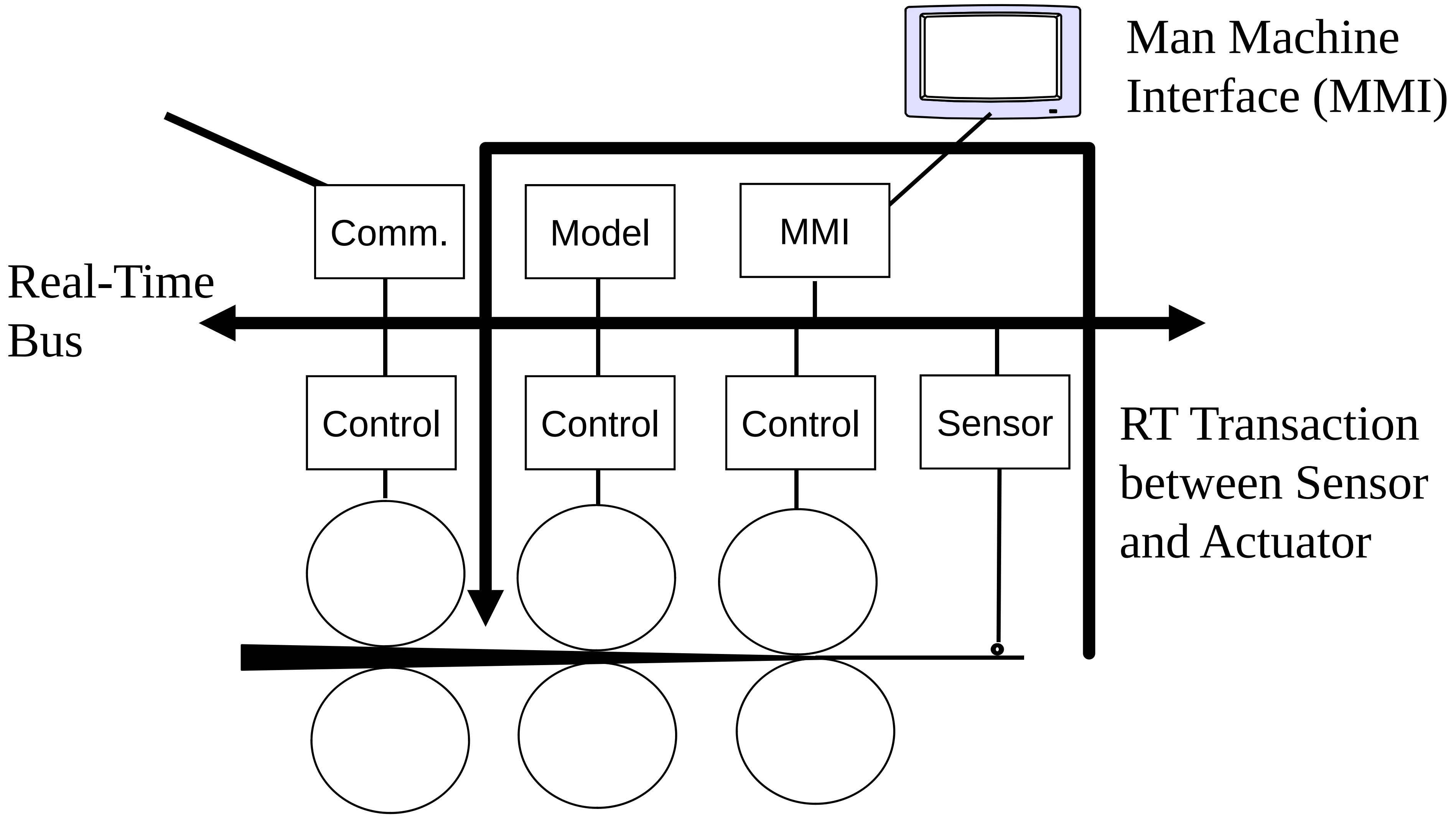
“The traffic light is green”

temporally accurate ?

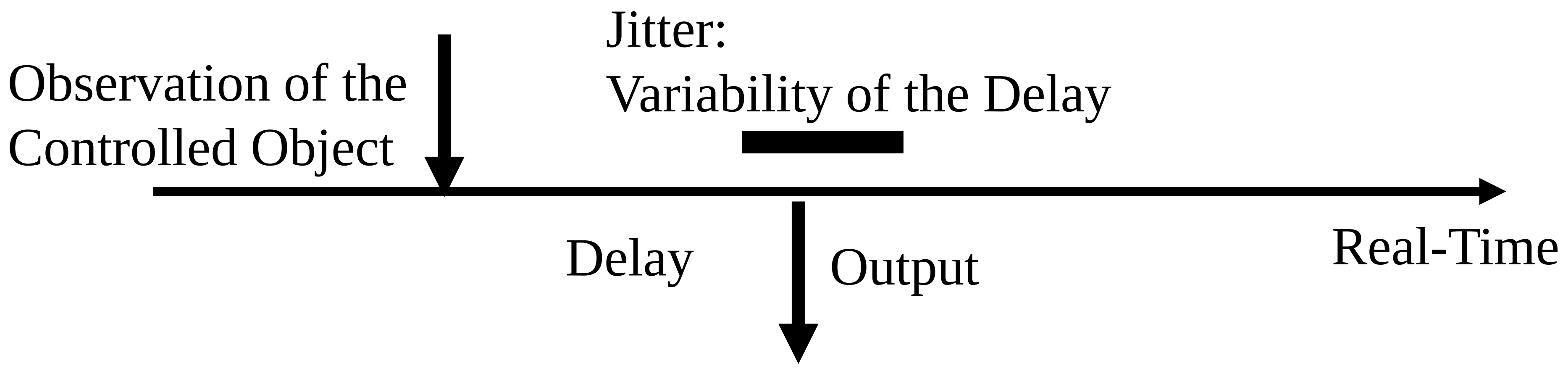


Temporal parameters are associated with real-time data.

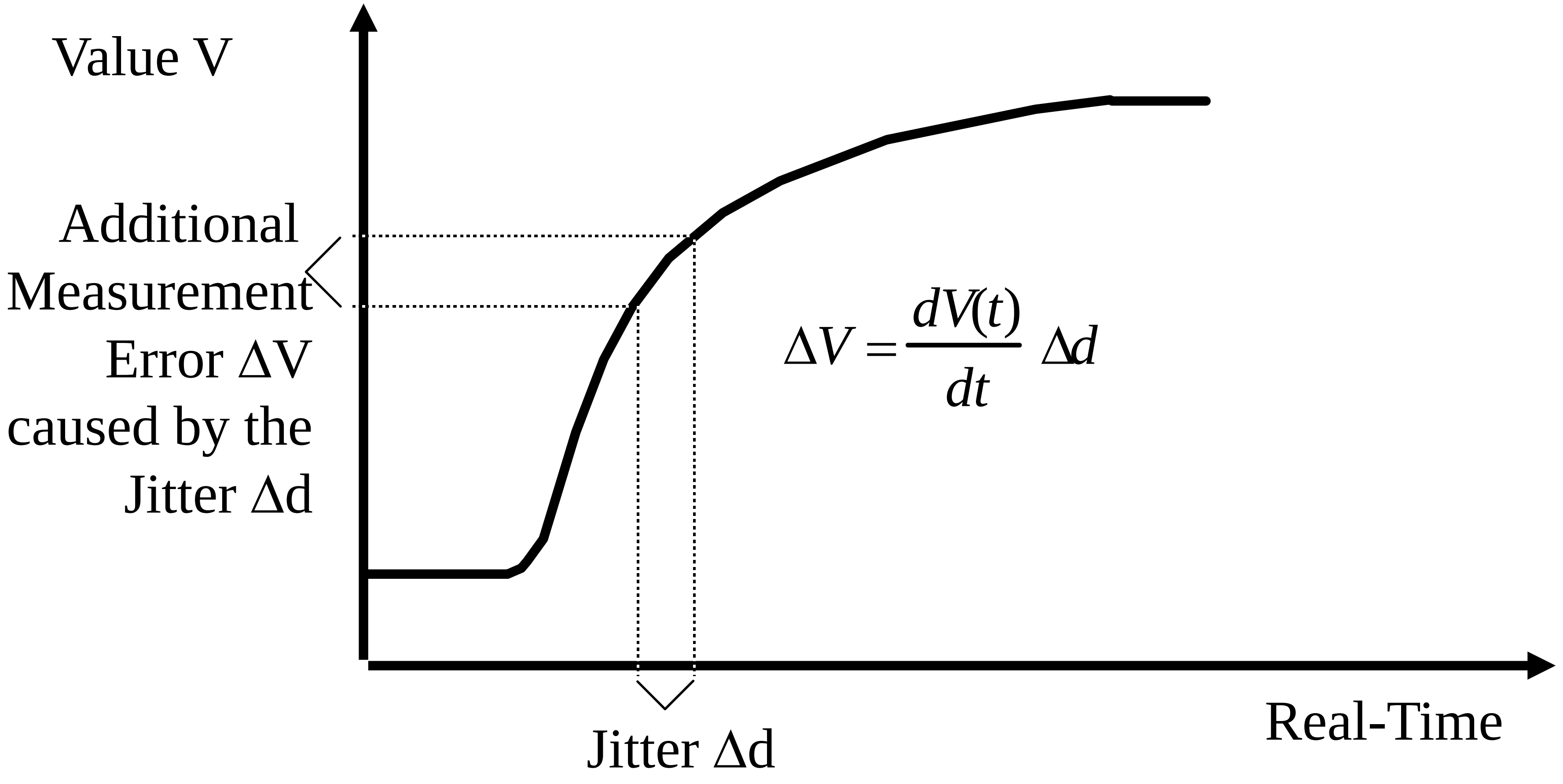
Real-Time Transaction



Jitter at the Application Level



The Effect of Jitter: Measurement Error



$$\Delta V = \frac{dV(t)}{dt}\,\Delta d$$

Jitter in Control Loops causes a degradation of control quality.

Protocol Execution Time in ET Systems

The execution time of an event-triggered protocol between two tasks depends on:

- the scheduling decisions by the operating system of the sender
- the buffer management of the sender
- the data link protocol
- the media access strategy
- the buffer management at the receiver
- the scheduling strategy at the receiver.

We call the maximum protocol execution time d_{max} and the minimum protocol execution time d_{min}.

ET Systems: Jitter at Critical Instant

A *critical instant* is a point in time, when all hosts in the ECUs try to send a message simultaneously. There is no phase control possible in ET system.

The message at the lowest priority level must wait until all higher priority messages have been sent (assume that all message have the same length).

Protocol execution time at critical instant (n ECUs):

$$d_{max} = n\, d_{trans}$$

Protocol execution time if the channel is free:

$$dmin = d_{trans}$$

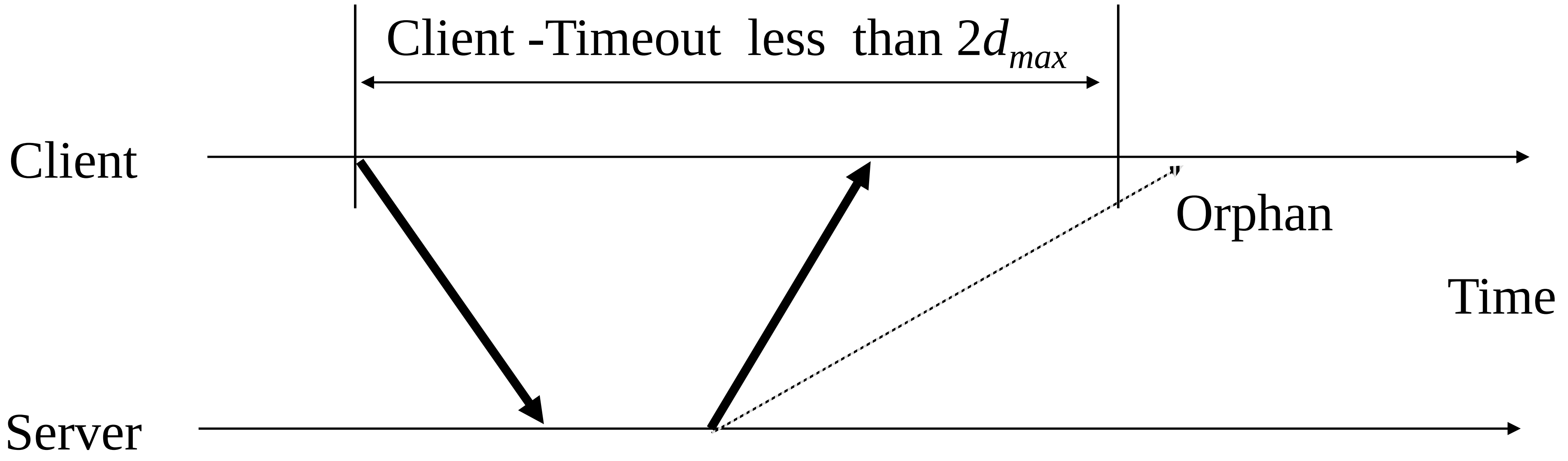
Jitter of the lowest priority message:

$$Jitter = (n-1)\, d_{trans}$$

The jitter depends on the number of ECUs in the system.

The Effect of Jitter: Orphans

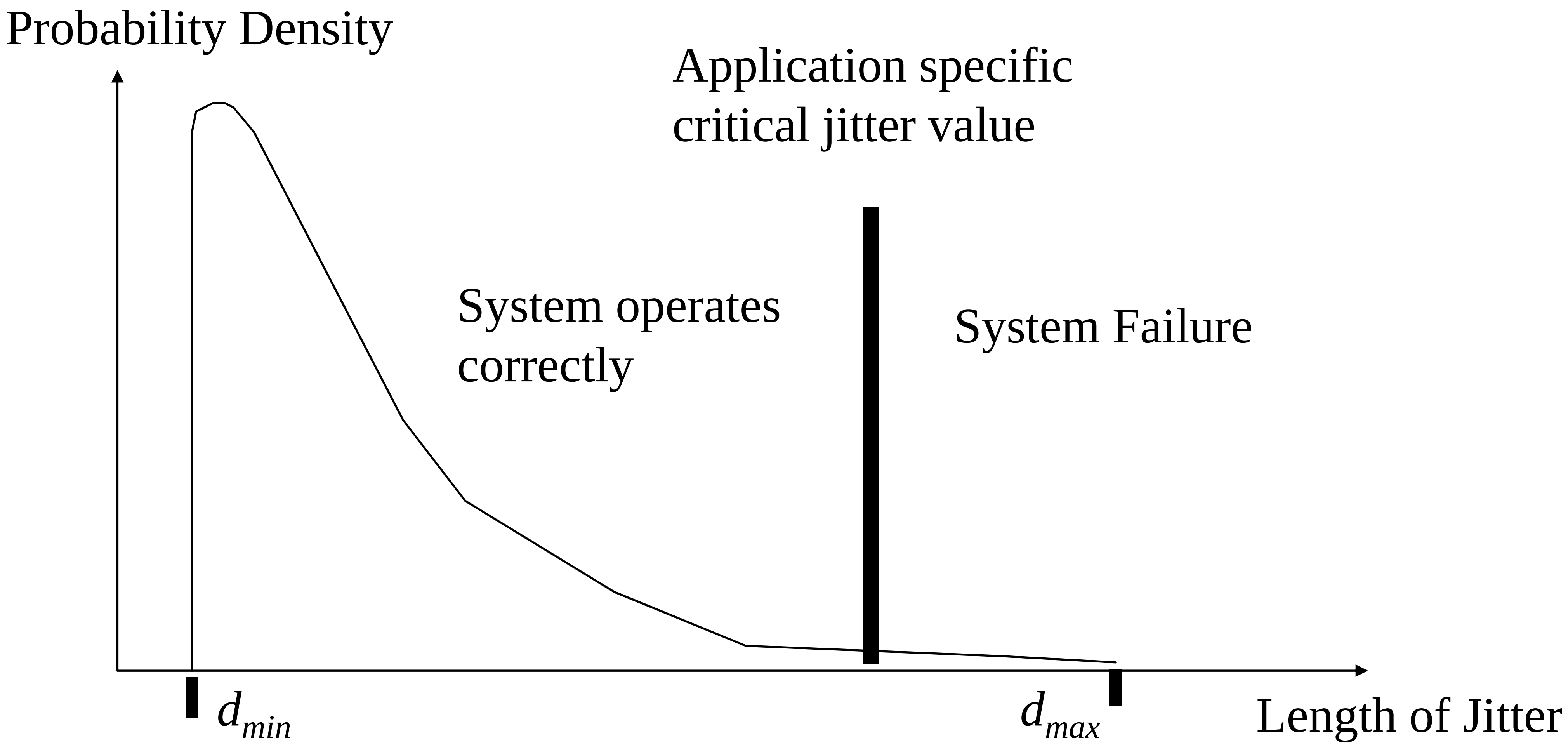
Request Response Transaction between a Client and a Server:



How large is d_{max} ?

It is not contained in the interface specification, available at the sub-supplier.

Probability of “Long” Jitter in ET Systems



Most of the time, the system will operate correctly.

Logical versus Temporal Control

The control scheme determines at what point in time the execution of a selected action will start. In RT systems it is necessary to distinguish between:

- ♦ *Logical Control* is concerned with the control flow within a task to realize the specified data transformation
- ♦ *Temporal Control* is concerned with the point in time when a task is to be started or when it has to be preempted by a more urgent task. Temporal control is closely related to scheduling

Synchronous Languages (e.g., Lustre) make a clear distinction between *logical control* and *temporal control.*

Rolling Mill Example

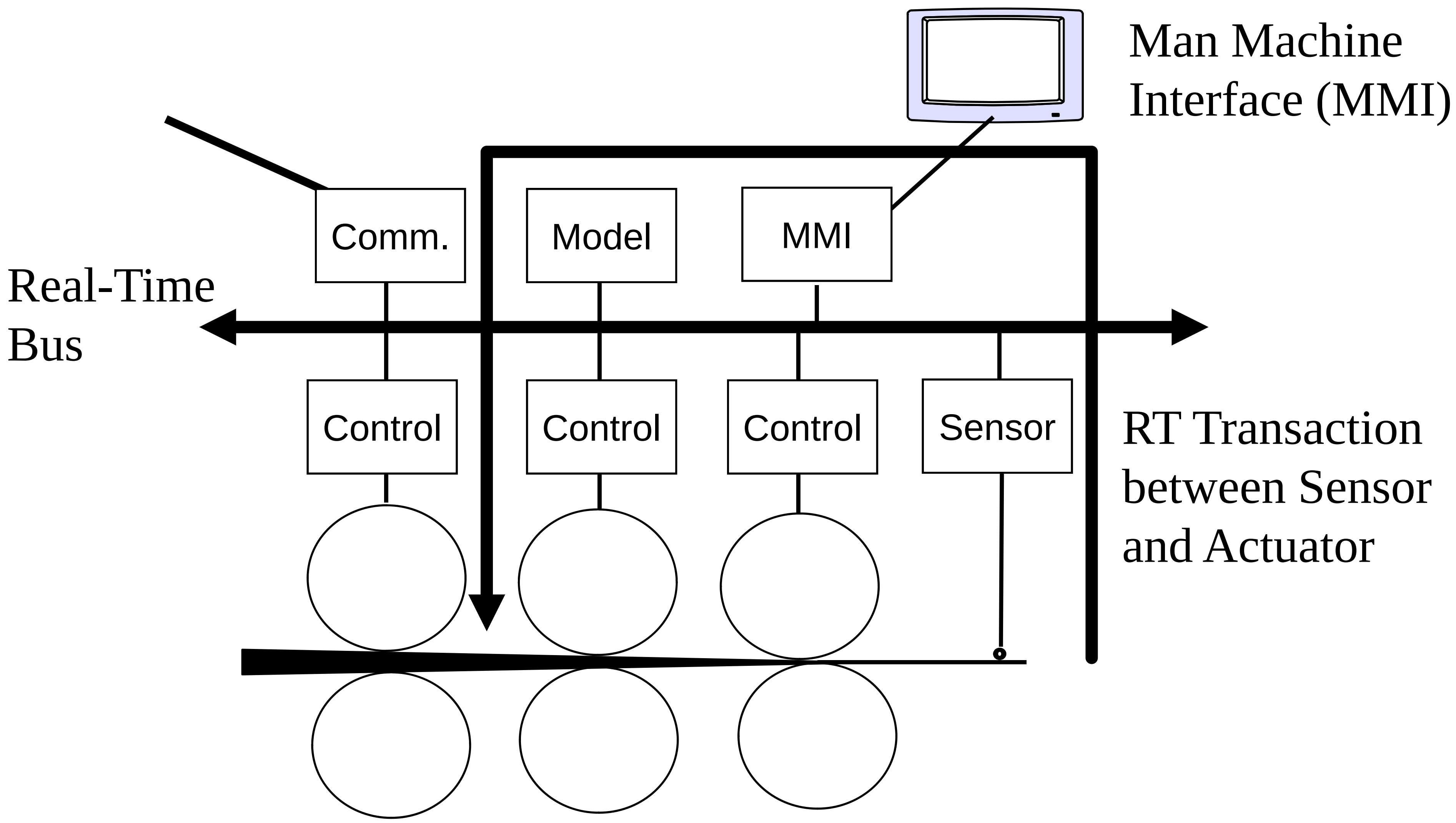
An alarm monitoring component should raise an alarm

WHEN $p_1 < p_2$ THEN raise alarm;

At a first glance, this specification of an alarm condition looks reasonable. A further analysis leads to the following open questions:

- ♦ What is the maximum delay in measuring p_1 and p_2 ?
- ♦ At what points in time must the alarm condition be evaluated?

Real-Time Transaction



Report on US Air Traffic Control

In February 1997, the United States General Accounting Office (GAO) published a report to the Secretary of Transportation, Mr.F. Pena, about the design and implementation of the new air traffic control system in the US.

The author of the report was Dr. R. B. Stillman, Chief Scientist for Computers and Telecommunications.

Principal Findings of GAO Report

- ♦ An architecture is the centerpiece of sound system development and maintenance.
- ♦ FAA is developing a logical architectural component for ATC modernization and evolution.
- ♦ FAA lacks a technical architectural component to guide and constrain ATC modernization and evolution.
- ♦ Without a technical ATC architecture, costly system incompatibilities have resulted and will continue.
- ♦ FAA lacks an effective management structure for developing and enforcing an ATC systems architecture.

What is a Technical System Architecture?

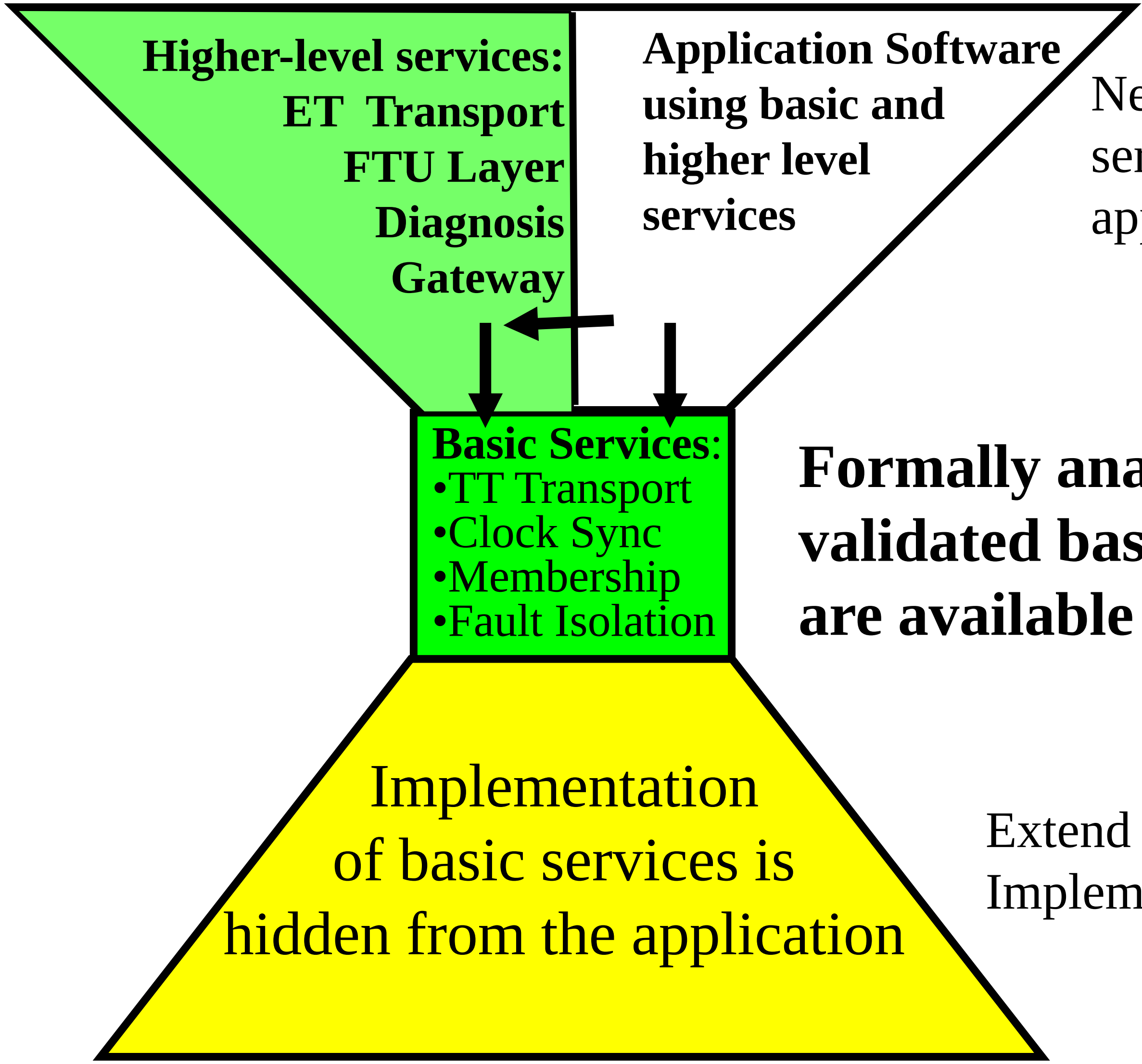
A *technical system architecture* is a framework for the construction of a system that constrains an implementation in such a way that the ensuing system is understandable, maintainable, extensible, and can be built cost-effectively.

Technical System Architecture (II)

- ♦ *Architectural style:* An architecture must provide rules and guidelines for the partitioning of a system into subsystems and for the design of the interactions among the subsystems.
- ♦ *Composability:* An architecture must provide a framework for the systematic construction of a system out of subsystems (components).
- ♦ *Property Match*: Components must comply with the *architectural style* to avoid a *property mismatch* at the component interfaces.
- ♦ *Elegance:* An architecture must *constrain* an implementation in such a way that the ensuing system is understandable, maintainable, extensible, and can be built cost-effectively--in other words, it is *elegant*.

Architecture Design is Interface Design

Waist-line Architecture (e.g., TTA)

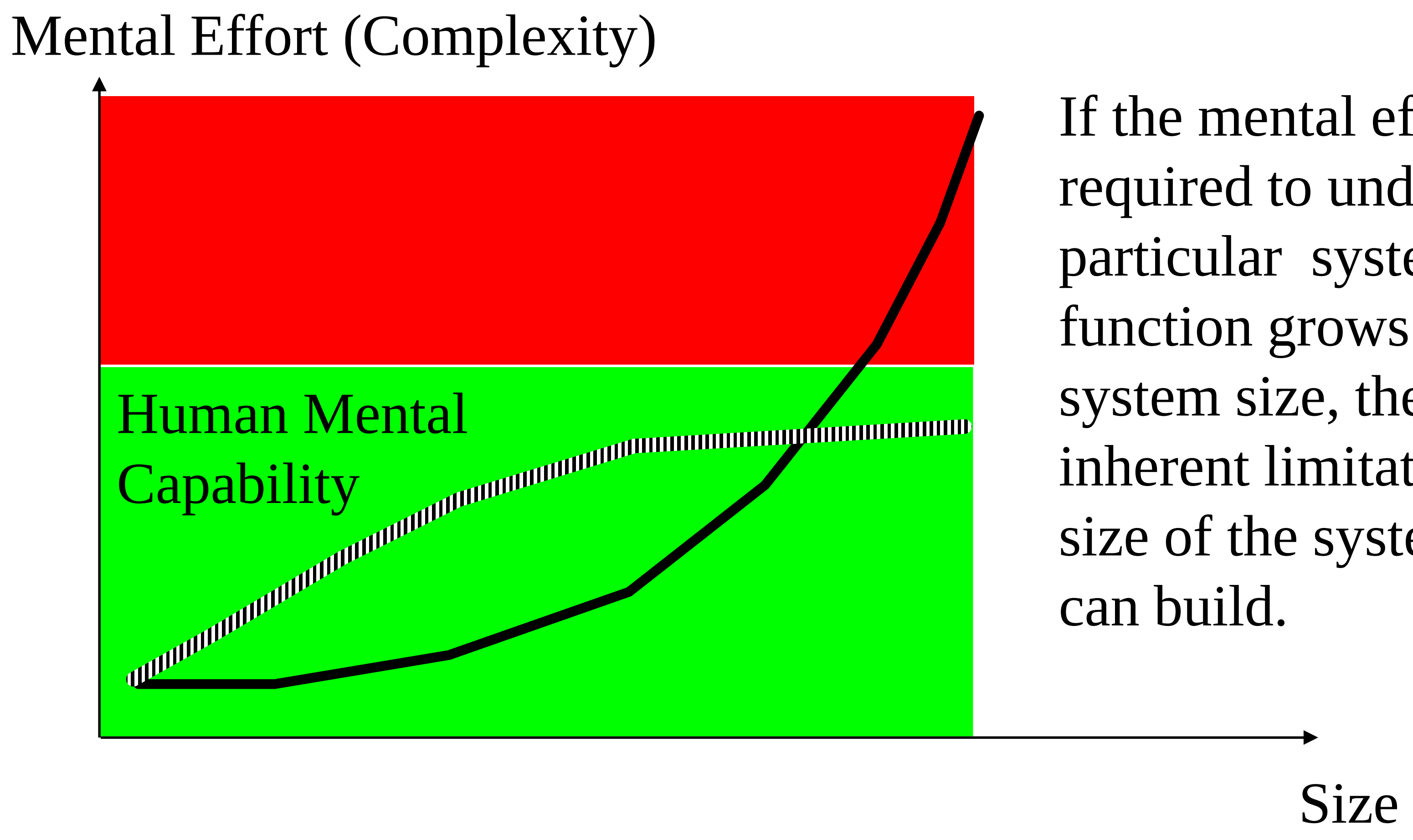


New high-level services to ease application development

Formally analyzed and validated basic services are available and stable

Extend the range of Implementation choices

Size versus Mental Effort to Understand



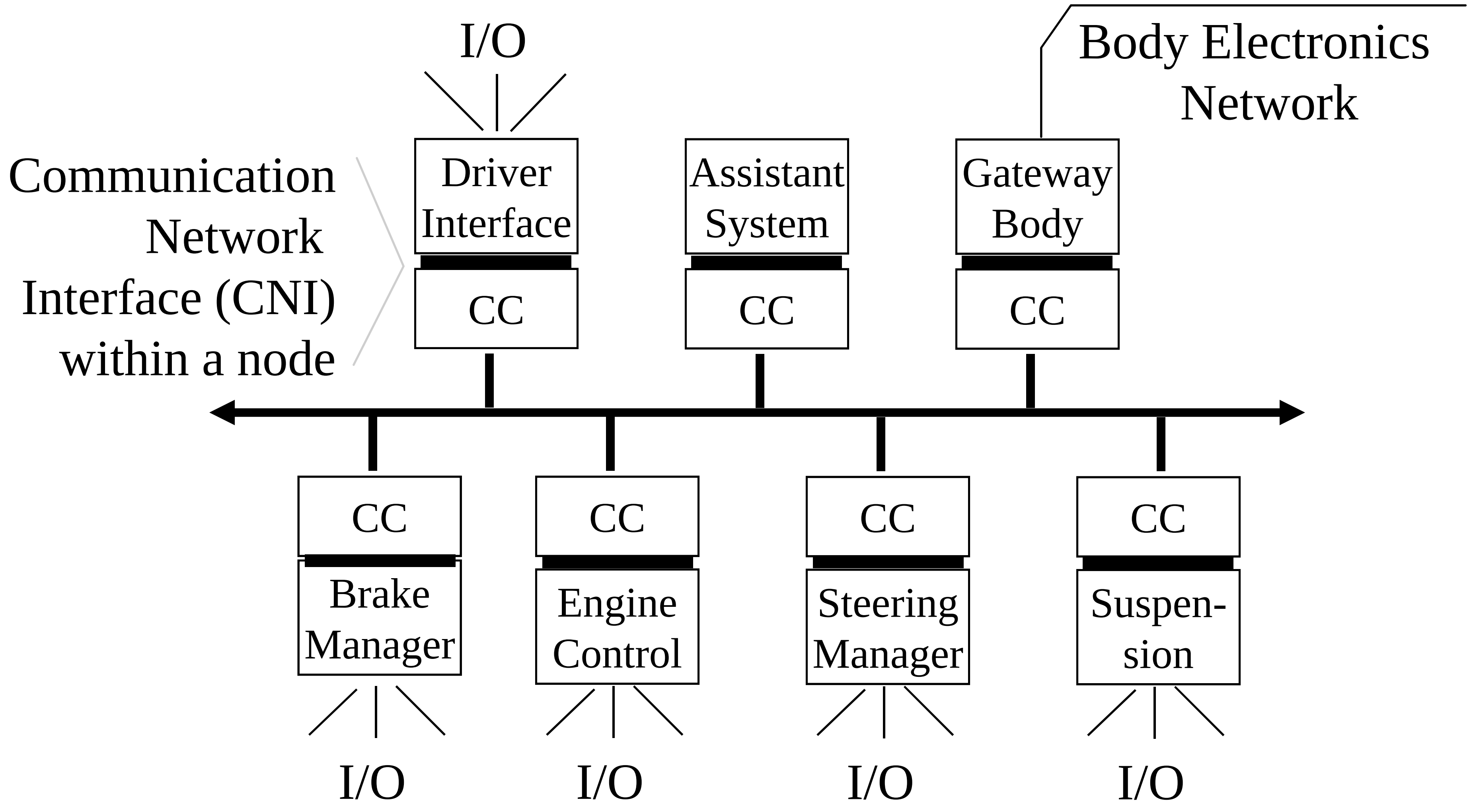
If the mental effort required to understand a particular system function grows with the system size, there is an inherent limitation to the size of the systems we can build.

Complexity and Size

- ♦ Large systems can only be built if the effort required to understand the system operation, i.e, the complexity of the system, remains under control as the system grows.
- ♦ The effort to understand any particular system function should remain constant, and should be *independent* of the system size.
- ♦ A large system contains many more different functions than a small system.
- ♦ The effort needed to understand all functions of a large system grows with the system size.

The design effort must be guided by technical system architecture.

Example of an Integrated Drive-by-Wire System



CC: Communication Controller

The Interoperability Problem–A Scenario

- ♦ A system integrator--e.g., an automotive company--specifies an Drive-by-Wire architecture and generates an **interface specification** for the subsystem (node) suppliers.
- ♦ The different nodes are developed by different sub-suppliers with respect to this interface specification.
- ♦ The acceptance tests of the nodes, as performed versus the interface specification, are o.k..
- ♦ The automotive company integrates the nodes into the system context and observes sporadic failures.

What is the cause of these failures and who is responsible for correcting these failures?

System Integration

Given a set of subsystems that are integrated to form a system.

A subsystem is a self-contained system, including hardware, software, and its own autonomous control, that provides a specified service to its environment.

At the system level, we distinguish between two types of services:

- ♦ **Prior Services**: The sum of the services that are provided by the isolated subsystems prior to the integration.
- ♦ **Emerging Services**: New services that come into existence by the integration

Composability

The prior service of each subsystems S_i can be characterised by prior properties P_{ik}. Assume that a given prior property P_{ik} of subsystems S_i has been established at the subsystem level before the integration.

An architecture is said to be *composable* with respect to this property P_{ik} if this property also holds at the system level after the integration.

Examples of such properties are:

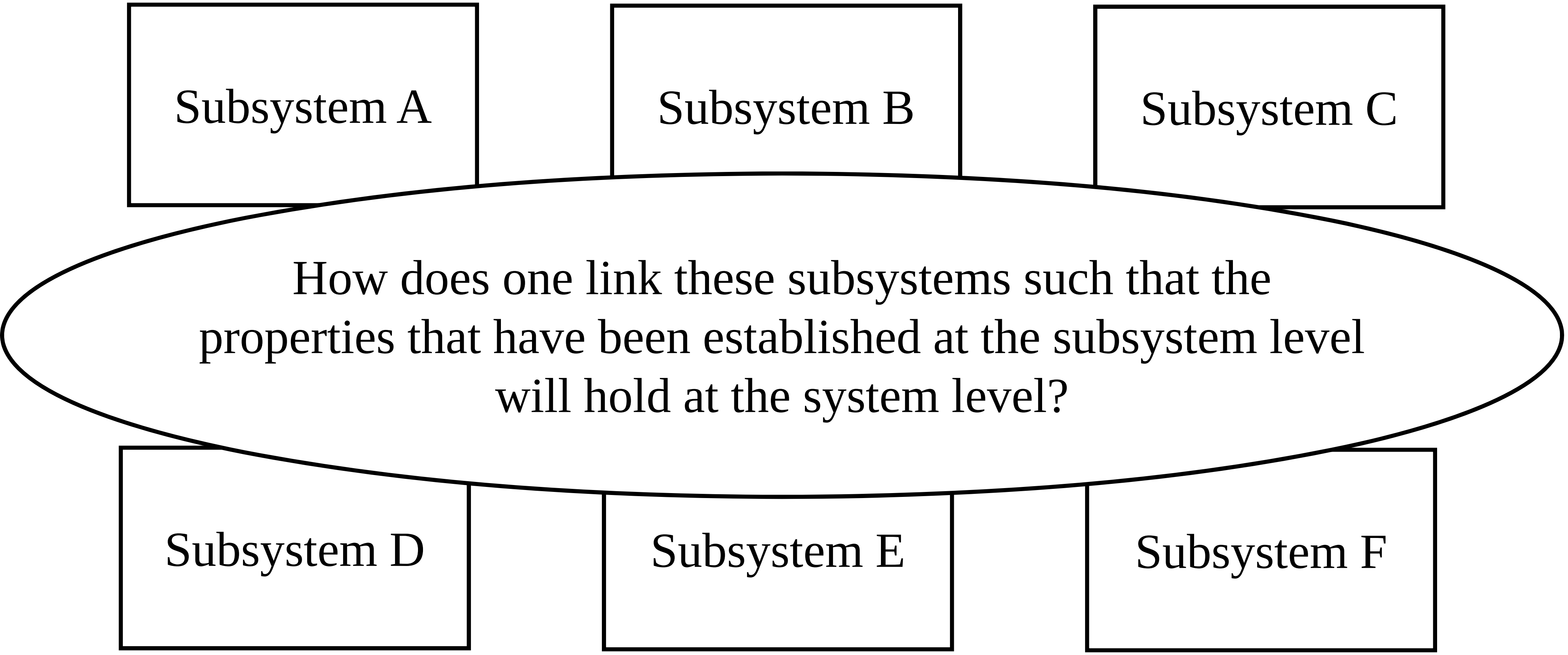
Timeliness

Testability

Composability ensures the stability of prior properties.

Composability: The Role of the Network

"The network is the only mechanism suitable to enforce and manage real-time operation of distributed systems" (Caro 1998) .

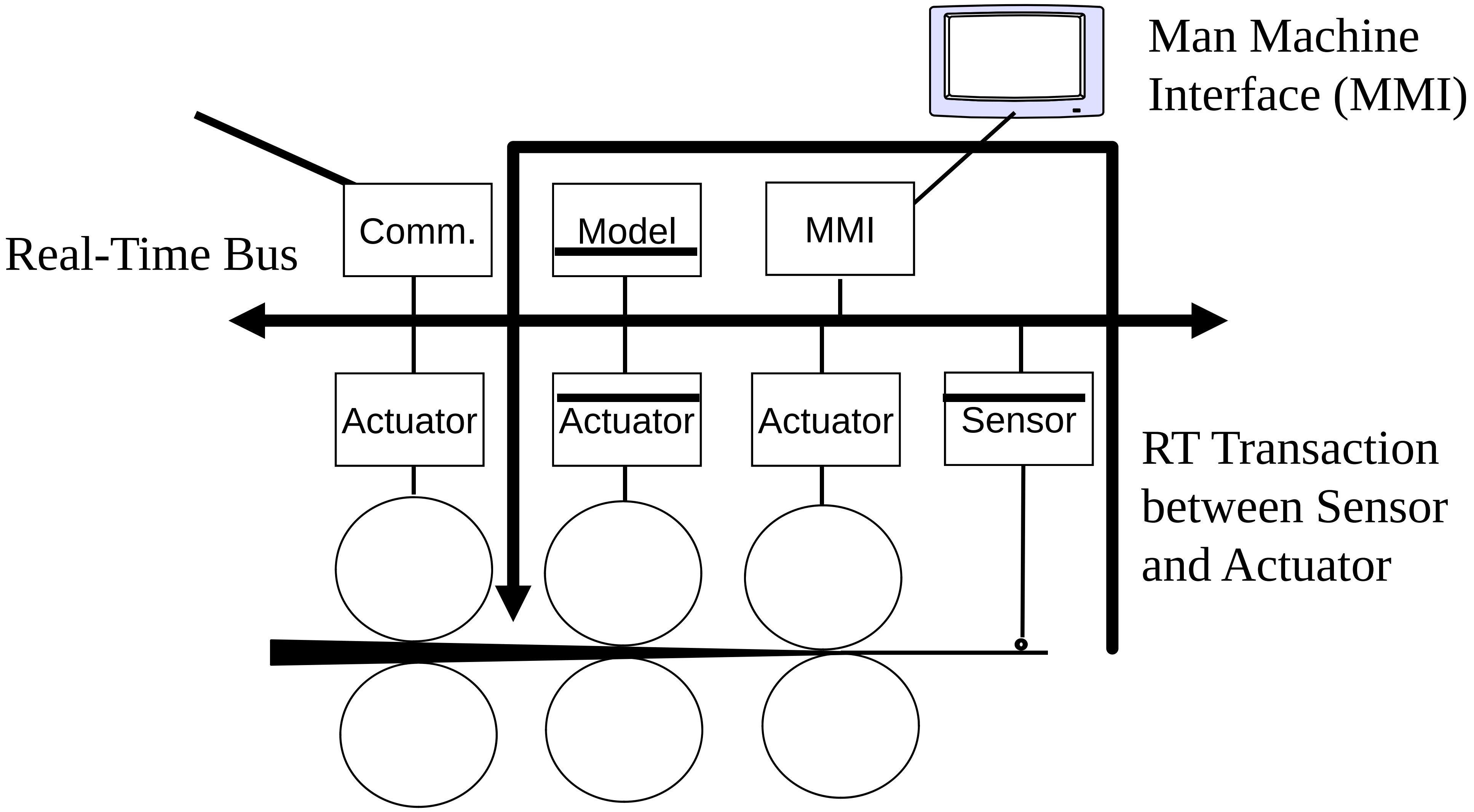

Subsystem A
Subsystem B
Subsystem C
How does one link these subsystems such that the properties that have been established at the subsystem level will hold at the system level?
Subsystem D
Subsystem E
Subsystem F

Principles of Composability

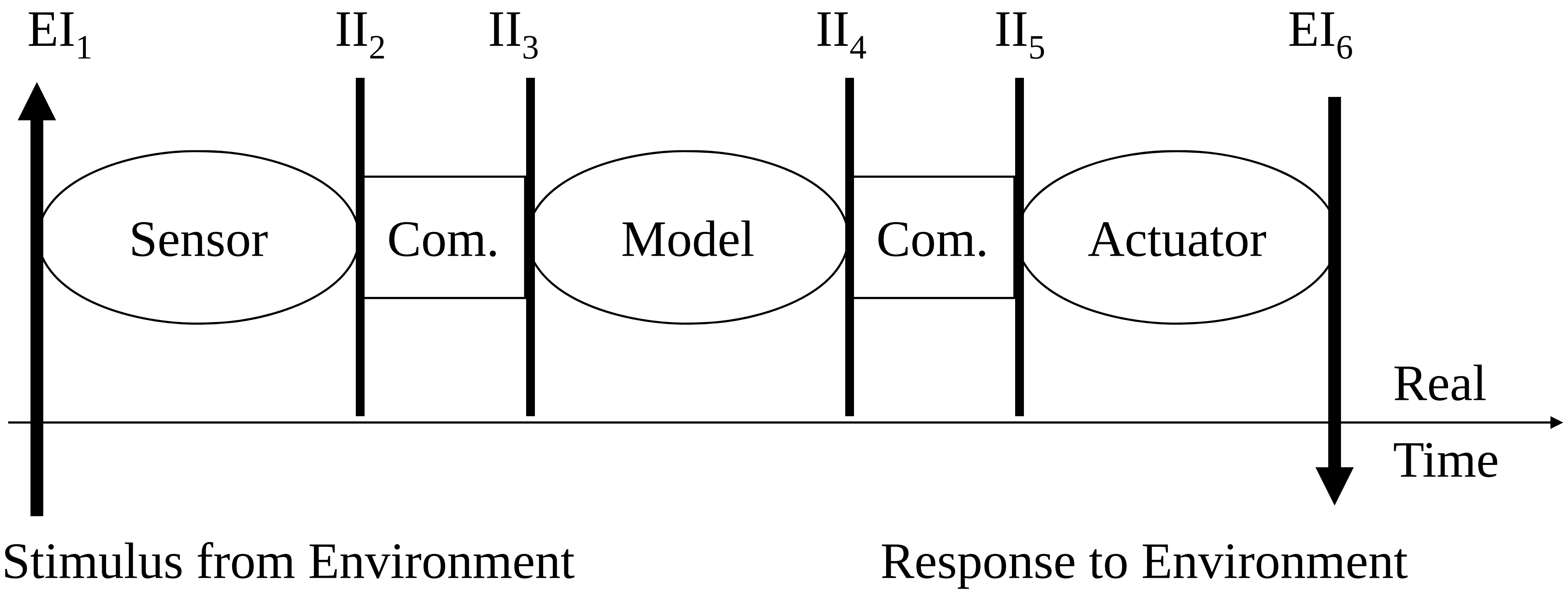
- **Independent development of components**--relates to the architecture support for a *two-level design process*:
  - architecture design with precise interface specification
  - component design, w.r.t these interface specification
- **Stability of prior services**--relates the components that are used in different system contexts (Solution of the reuse problem).
- **Non-Interference**--component integration should be linear and not circular--relates to the communication system.
- **Fault Containment**--Components should form Fault-Containment Units

Furthermore, if fault-tolerance is to be implemented by component replication, the component must be ***replica deterministic***.

An Example: Rolling Mill


Man Machine Interface (MMI)
Comm.
Model
MMI
Real-Time Bus
Actuator
Actuator
Actuator
Sensor
RT Transaction between Sensor and Actuator

Real-Time Transaction

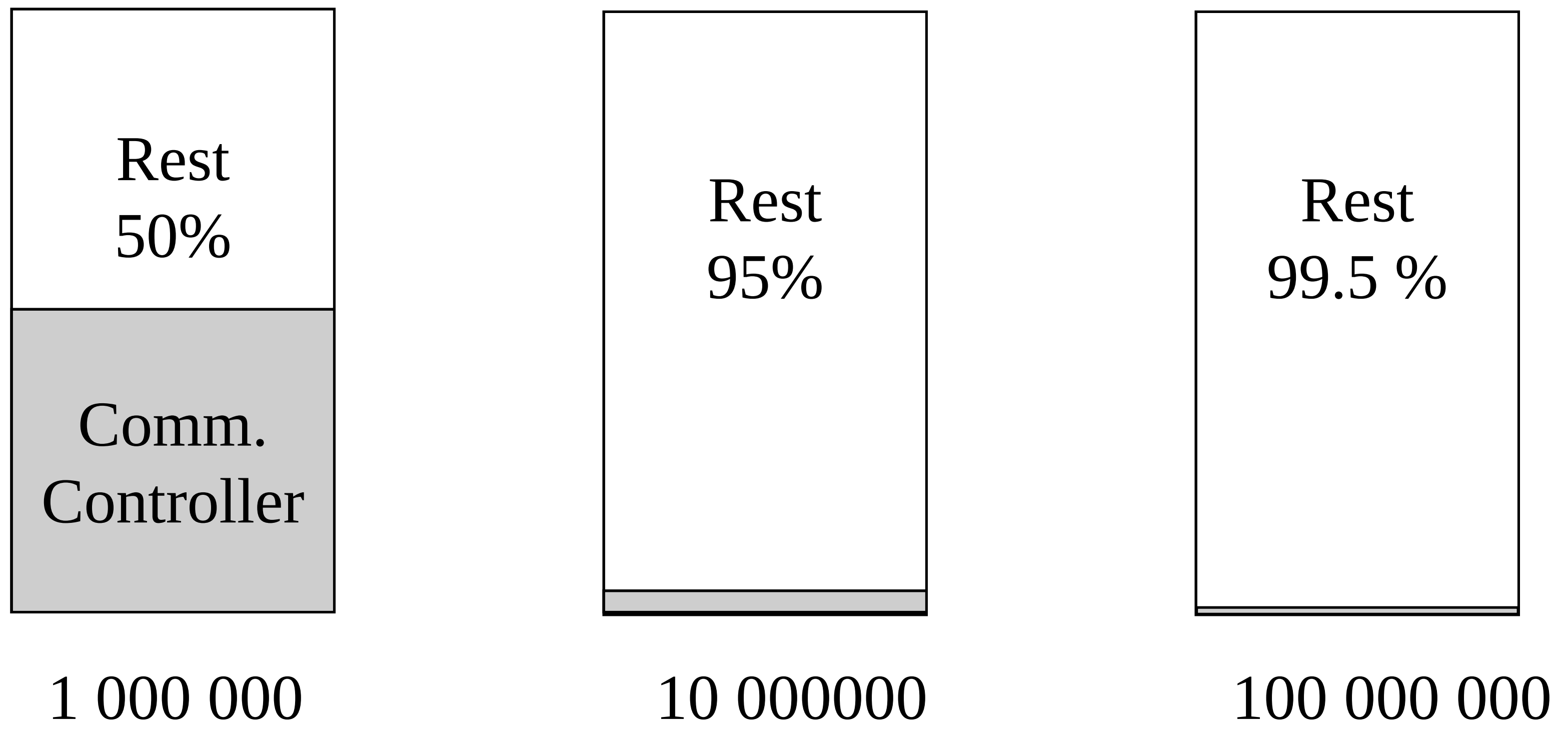


Technology Trend to Distributed Systems

- **System on a Chip (SOC):** A complete computer system, including, CPU, Memory, I/O, Communication Controller, Operating Systems, and Application Software can be implemented on a single silicon die: e.g., IBM-Sony-Toshiba Cell
- **Smart Sensors**: Sensing Element, signal processing, calibration, diagnosis, communication control on a single die (MEMS).
- **On-Chip Oscillators** for low-cost nodes: cheap, but imprecise
- **COTS**: Commercial off the shelf components comprising hardware and software
- **Integrated Fault Tolerance**: to mask faults, e.g. SEU (single event upsets)

Cost of Distribution

The cost of distribution decreases with increasing VLSI integration.
Let us assume, the on-chip implementation of a complex communication controller consumes the equivalent of 500 000 transistors:



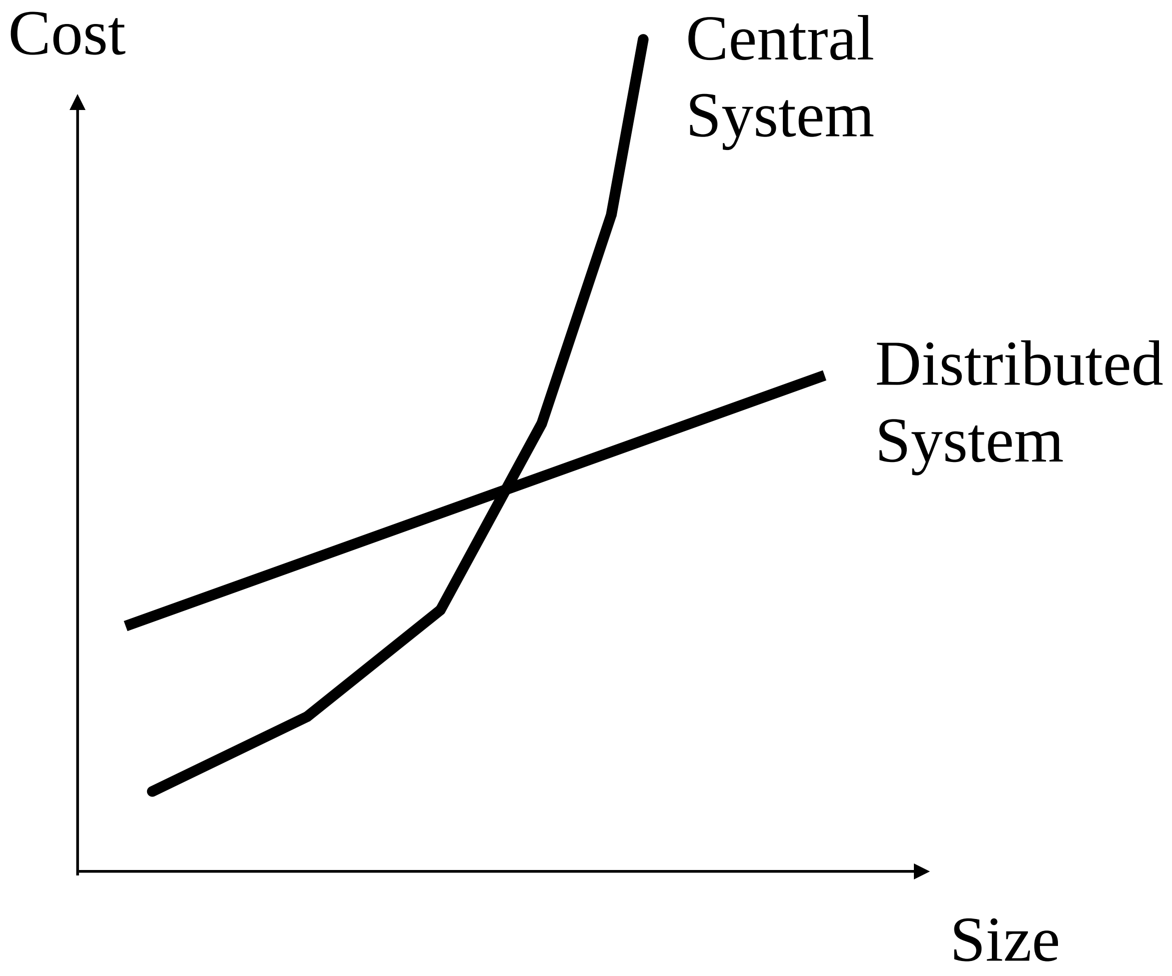
Total Numbers of Transistors on a Single Die

Silicon Die Costs

According to Hennesy and Patterson (p.60), the

Cost = f (Die-area^{3})

On the other hand, a distributed system requires more packaging and an additional die area for the communication controller.



Hardware Developments--Moore’s Law

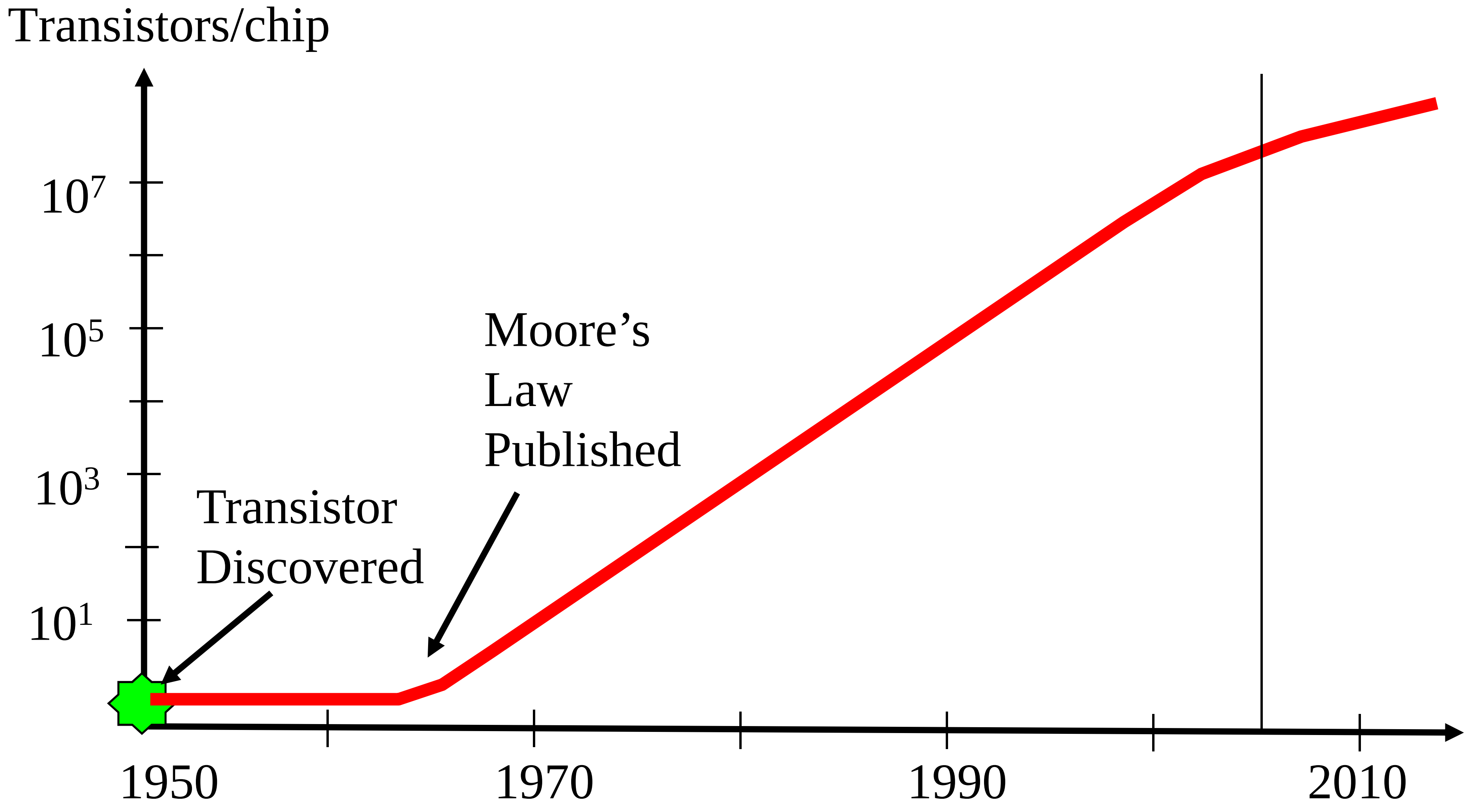
During recent history, we have seen tremendous advances in the field of computer hardware:

- Moore published his *law* of exponential growth of semiconductor parameters (e.g., device density doubling roughly every 18 months) in 1965. About 25 generations of hardware give us today (2004) dies with more than 500 Million transistors--we expect more than one billion transistors per die before the year 2010.
- Examples:

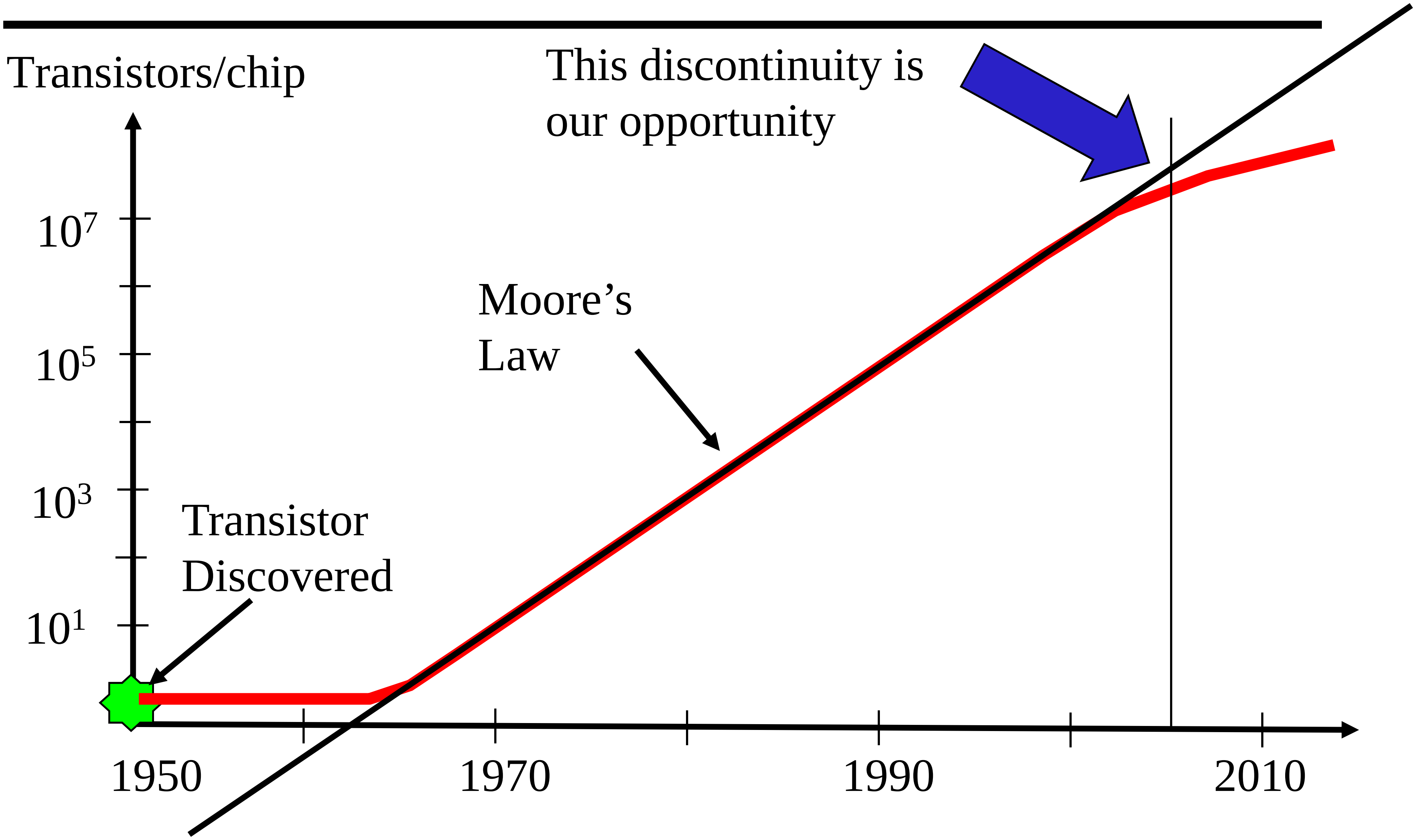
Risk II 32 bit 40 k transistors:	1983	3 000 nm	60 mm2
	2004	90 nm	0.05 mm2

- Moore’s law in its *present form* is coming to end.
- Every discontinuity is an opportunity.

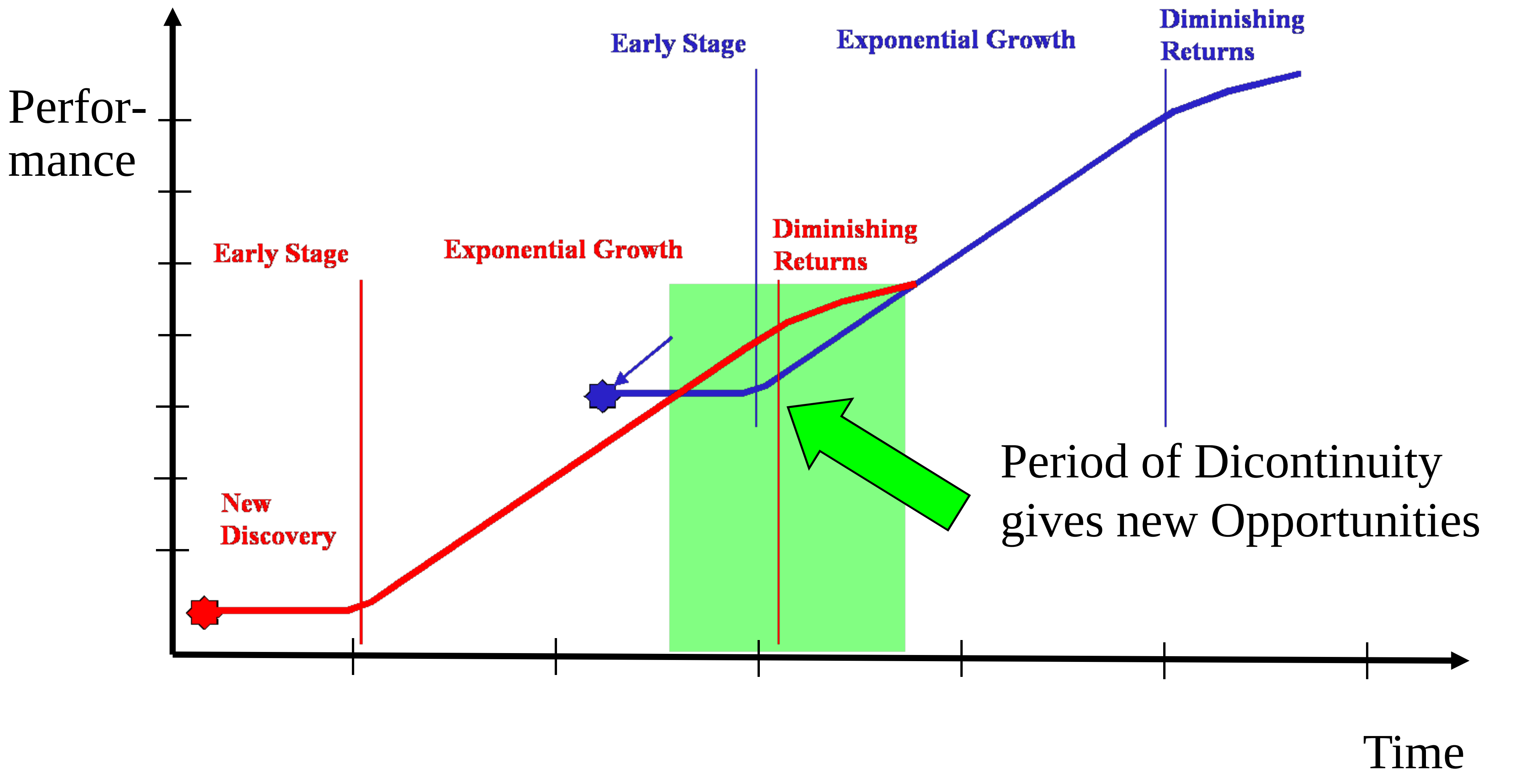
S-Curve of the Semiconductor Industry



S-Curve of the Semiconductor Industry



S-Curves often Overlap



Discontinuities in Computer Technology

We are approaching three walls:

- ♦ **Die Size Wall:**
  Die sizes are not increasing any more (as originally predicted by Moore's law)
- ♦ **Silicon Technology Wall:**
  Moore's law approaches its limits when it comes to the further shrinkage the devices on a die
- ♦ **Architecture Wall:**
  It becomes more an more difficult to achieve a linear speedup of performance at the architecture level

CMOS Transistor Performance Reaches Limits

The International Roadmap on Semiconductors (ITRS) 2003 has identified a number of scaling barriers for the next generation of semiconductors.

- Power density (Pentium 4: 80W/cm^2 vs. Stovetop: 10 W/cm^2)
- Relation of passive power to active power dissipation
- Gate oxide tunneling
- Interconnect-wire width
- Transient failure rate
- Fabrication cost (mask)

There is no alternative to the silicon CMOS transistor in sight.

Architecture Wall: Latency Lags Bandwidth

Relation latency/bandwidth improvement over the past 20 years

Patterson CACM 10, 2004

	CPU	DRAM	LAN	DISK
Bandwidth increase/yr	1.5	1.23	1.39	1.28
Latency (1.2 -1.4/yr)	1.17	1.07	1.12	1.11
Time to double Bandwidth (yr)	1.7	2.9	2.1	2.8

Over the past twenty years, bandwidth has improved by more than the square of the latency.

Example: It becomes more and more difficult to match processor performance and memory performance: introduction of caches, *that do no useful work.*

Pollack's Rule

Pollack's Rule: 1.4 performance increase for every doubling of transistor count. If *performance per transistor* is the metric, than 1/1.4 *performance loss per transistor* for each generation--Performance loss over 25 generations more than 1000.

If we include software, the performance loss factor is substantially higher.

Example: To overcome the memory latency the *Intel Itanium* uses three level of caches, where 85 % of the 211 000 000 transistors can be found.

Solutions to Overcome the Architecture Wall

- ♦ Tiled Architecture--Multicore
  Provide many general purpose processors in a tiled (regular) fashion to solve regular problems with high parallelism (e.g., matrix inversion).
- ♦ Function-Block Architectures (FBA)
  Unload the general purpose processor and solve well-defined functions by dedicated hardware units.
- ♦ Direct Mappling of Algorithms into FPGAs
  Map algorithms directly into FPGAs, using the full parallelism of this technology--avoid the van Neumann bottleneck.

Systems on Chip:

Current Semiconductor Technology makes it possible to design a selfcontained 32 bit computer system, including 1 Mbyte of memory, Network Access and I/O on a single die, e.g., the Motorola Golden Oak Chip.

Development cost of an SOC: > 10 Mio US $

Production cost: < 10 US $

Number of devices sold (in millions)	0.1	1	10
Development overhead (per device)	1 000%	100%	10 %

Example: Cell Processor announced 2005

Joint development of IBM, Sony and Toshiba, 90 nm process 250 Mio Transistors, 221mm^2 , 4 Ghz, 250 GFLOPS

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

SPE Processor

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Specialized Processor for SIMD-type data streams

256 Kbytes of private memory (LS0-LS3), access latency is 6 cycles

32 bit instructions, 128 registers

Ca 15 mm^2

BIU less than 1 mm^2

Function-Block Architectures (FBA)

- Possible Function Blocks
  - Graphic Processing Unit (GPU)
  - Communication Protocol Hardware
  - Security Mechanisms
  - Fault-Tolerance Layer
  - Connector Units
- Example of an FBA: The DECOS Architecture
  In the DECOS intergrated project, sponsored by the EU IST program, an integrated function block architecture is developed for dependable distributed real-time systems.

Field Programmable Gate Arrays (FPGA)

1991-2004 progess of FPGAs:
speedup 40x, power per function 1/50, size 200x, cost 1/500

In the next five years FPGA improvements (3x in 3 yrs) are expected to outpace improvements in general purpose processors (2x in 3 years).

Example 2004: Virtex 4 from Xilinx: 90 nm technology, 500 million transistors, 450 Mhz Power PC(.9mW/Mhz) 1 gigabit I/O on any pin

Speed-up example: Fractal Calculation

Processor	GFLOPS	Power	Power/GFLOP
Intel Xeon 1.8	0.17	40	0.004
Virtex II FPGA (Xilinx)	24	20 W	1.2
Relation	140	0.5	300

Cump, C. et al, Proc. HPEC 2004, p.63

FPGA Challenge

Technology is giving us a *world of gates* (hundreds of millions) that are implemented on a single die

- ♦ We must develop algorithms and design methods and tools that map an algorithmic solution directly to this *world of gates* taking advantage of the inherent parallelism of the execution environment.
- ♦ Fast dynamic reconfiguration of the hardware base is possible in support of mode changes.
- ♦ Logic of the hardware is *uncommitted*, therefore *the cost-benefit of mass production can be fully utilized.*
- ♦ The only limitation we see today is the limited design methodology and the limited ASIC expertise of design teams.

How Good are the Predictions?

Eight Edition of the SIA Roadmap will be published at the end of 2007--First edition published in 1993

	Predicted in 1992 for 2007	Actual 2007
Feature Size (nm)	100	65
Chip size (mm^2)	1000-1250	150-750
Clock speed (Ghz)	1	4-5

Worldwide Semiconductor Market in 2007 about 250 Mia $

Source: Chris Mack, Semiconductor Magazine, 9/1/2007

ITRS Predictions 2007

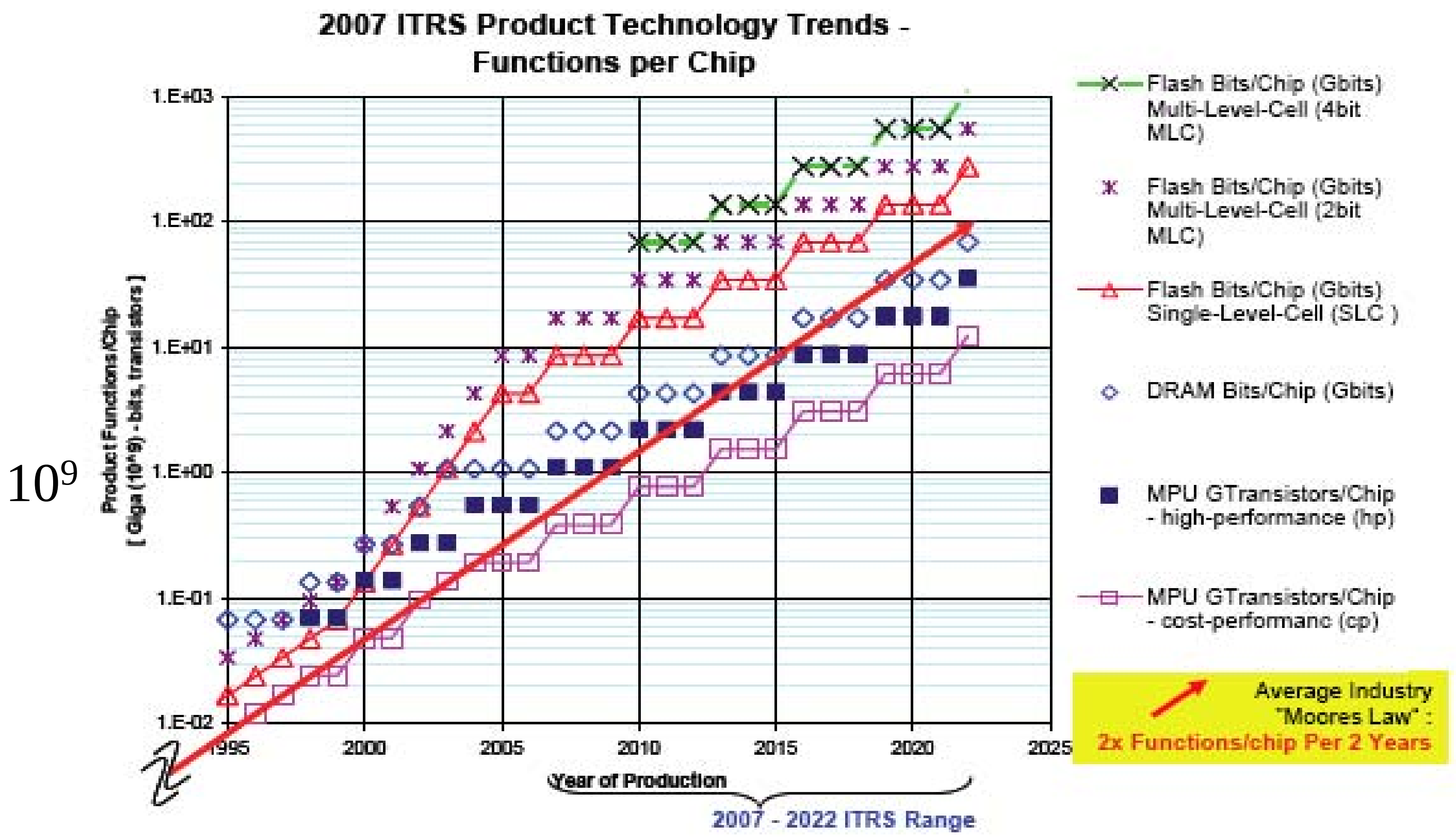


Figure 9 2007 ITRS Product Technology Trends: Product Functions/Chip and Industry Average "Moore's Law" Trends

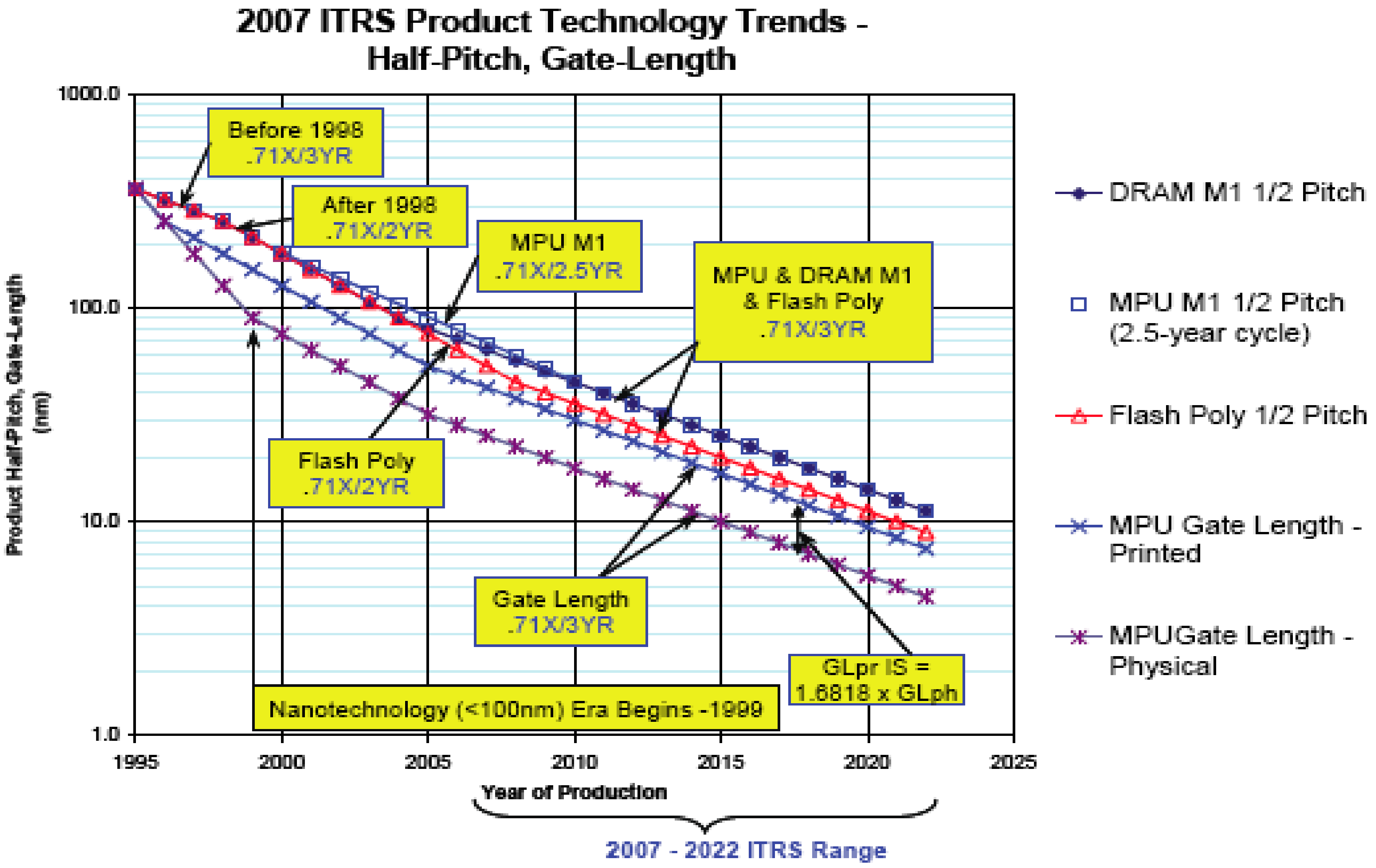


Figure 7 2007 ITRS—Half Pitch and Gate Length Trends

State of Silicon Technology in 2007:

- ♦ 65 nm feature size in production
- ♦ Asymptotic production cost of CMOS between 2 and 5 €/ cm^2
- ♦ SRAM: 1000 Million transistors/cm^2
- ♦ Logic: > 200 Million transistors, 80 million gates
- ♦ 5 μc/gate
- ♦ Today it is possible to implement a 32 bit microprocessor in less than 0.1 mm^2 of silicon at a production cost of less than 0.3 cent.
- ♦ Production runs: Many more than one million devices must be produced to come *within 10% of the asymptotic production* cost (design and mask costs).

Estimated Parameters of an SoC around 2025

	2007	2025
Feature Size (nm)	65	<10
DRAM Mbits/mm^2	40	>500
SRAM Mbits/mm^2	1	>10
Million transistors/mm^2	10	>50
Chip size mm^2	200	200
Frequency in GHz	4	>25
Cost/ mm^2 (in cents)	10	10
Cost per transistor (μcents)	1.2	.12
Cost per CPU (200k--cents)	0.5	0.04
MTTF/chip permanent (years)	1000	100
MTTF/chip transient (years	1	<0.01

Power Budget

- ♦ Indoor Devices (Desktop, Multimedia)
  Power Budget: 1 Watt - 100 Watt,
  Energy Source: Mains
- ♦ Normadic Devices (Mobile Phone, I-Pod)
  Power Budget: 100 mW - 1 Watt
  Energy Source: Battery
- ♦ Transducer Devices (Sensor Nets)
  Power Budget: 100 µWatt - 100mWatt
  Energy Source: Ambient

Ref: Lauwereins, Imec, MSOP 2006

Power Consumption of Chips

Power Dissipation is a new Design Problem: in modern Microprocessors 50-75 Watt/cm^2

Static Power:

Power consumed when the device is powered up but no signals are changing value.

Dynamic Power:

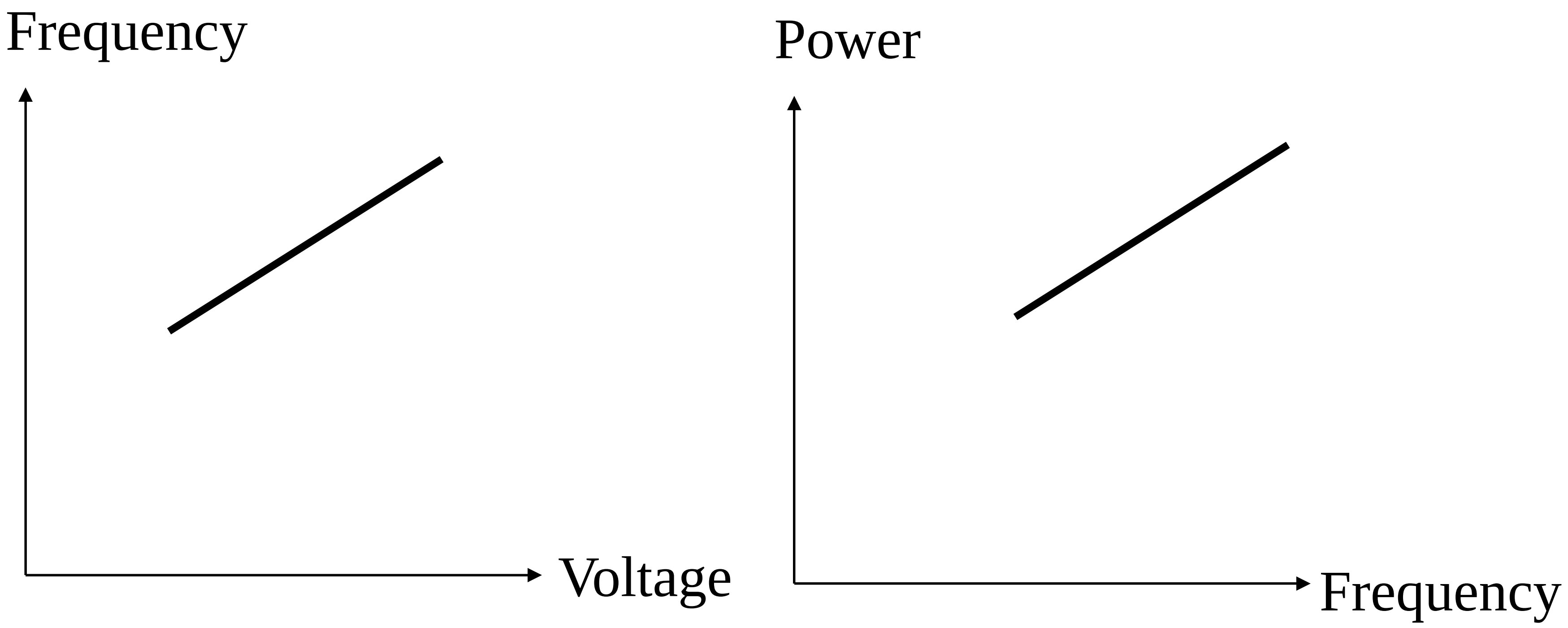
The power consumed for the activity when the device is active:

$$P_{dyn} = C_{eff} \bullet V^2 \bullet f_{clock}$$

P_{dyn}	*Dynamic Power consumed*
C_{eff}	*Effective Capacitance*
V	*Supply Voltage*
f_{clock}	*Clock Frequency*

Power versus Frequency

$$P_{dyn} = C_{eff} \bullet V^2 \bullet f_{clock}$$



Power Savings with Voltage and Frequency Scaling

Static vs. Dynamic Power

Node	90 nm	65 nm	45n
Dynamic Power per cm^2	1X	1.4X	2X
Static Power per cm^2	1X	2.5X	6.5X
Total Power per cm^2	1X	2X	4X

Source: Keating et. al. *Low Power Methodology Manual for System on Chip Design*, Springer, 2007, p.3

Power Gating vs. State Retention

Power Gating: Power gating consists of selectively powering down certain blocks in the chip while keeping other blocks powered up.

State Retention: The state that is stored in a powered-down block must be retained in the chip:

HW based with retention registers

SW based via explicit state saving

Example: Cell Processor

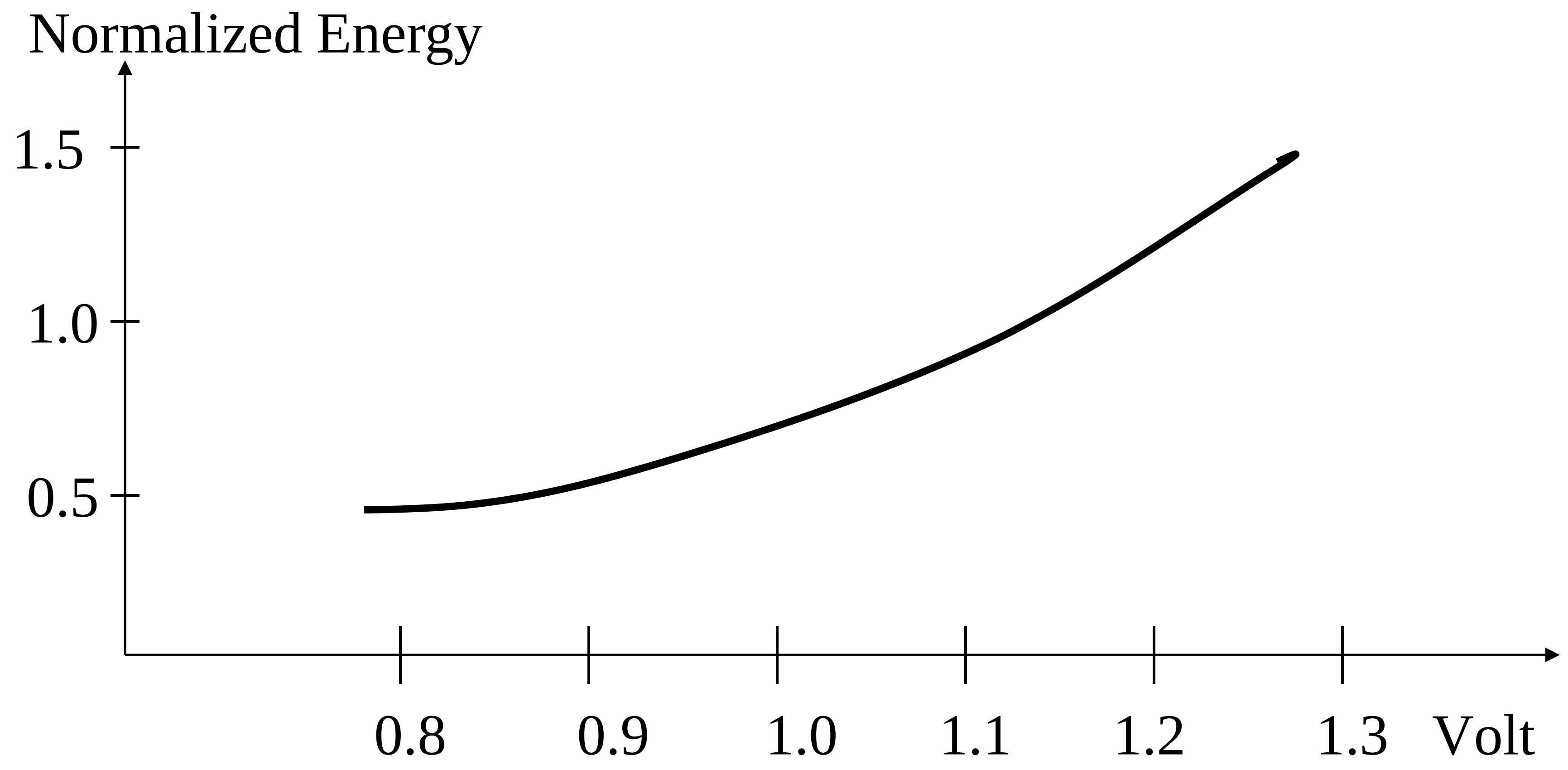
Joint development of IBM, Sony and Toshiba, 90 nm process 250 Mio Transistors, 221mm^2 , 4 Ghz, 250 GFLOPS

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

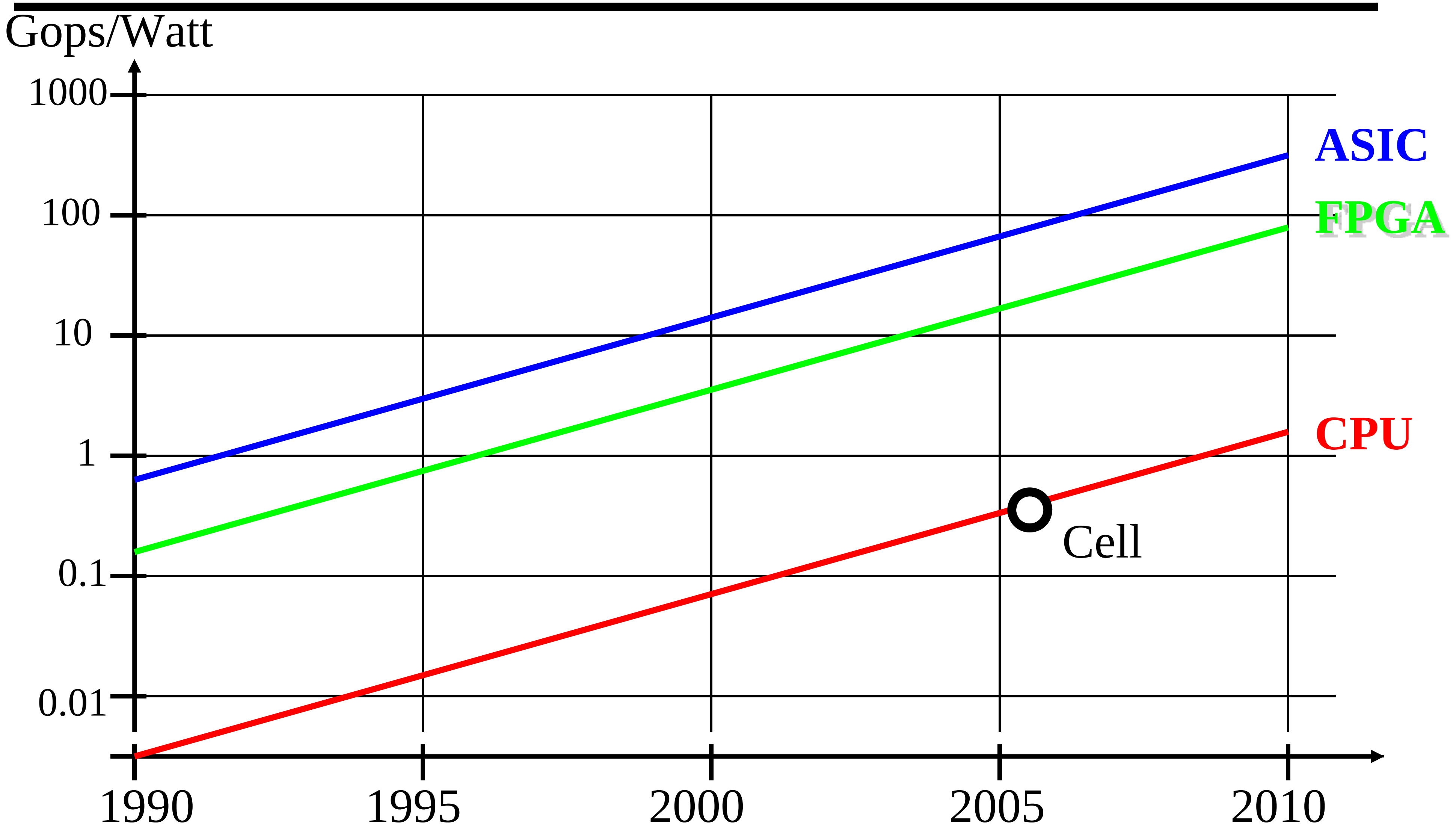
Consistent Notion of State

“The state enables the determination of a future output solely on the basis of the future input and the state the system is in. In other word, the state enables a “decoupling” of the past from the present and future. The state embodies all past history of a system. Knowing the state “supplants” knowledge of the past. Apparently, for this role to be meaningful, the notion of past and future must be relevant for the system considered.” (Taken from Mesarovic, Abstract System Theory, p.45)

Example: ULTRA 926 SoC Prototype

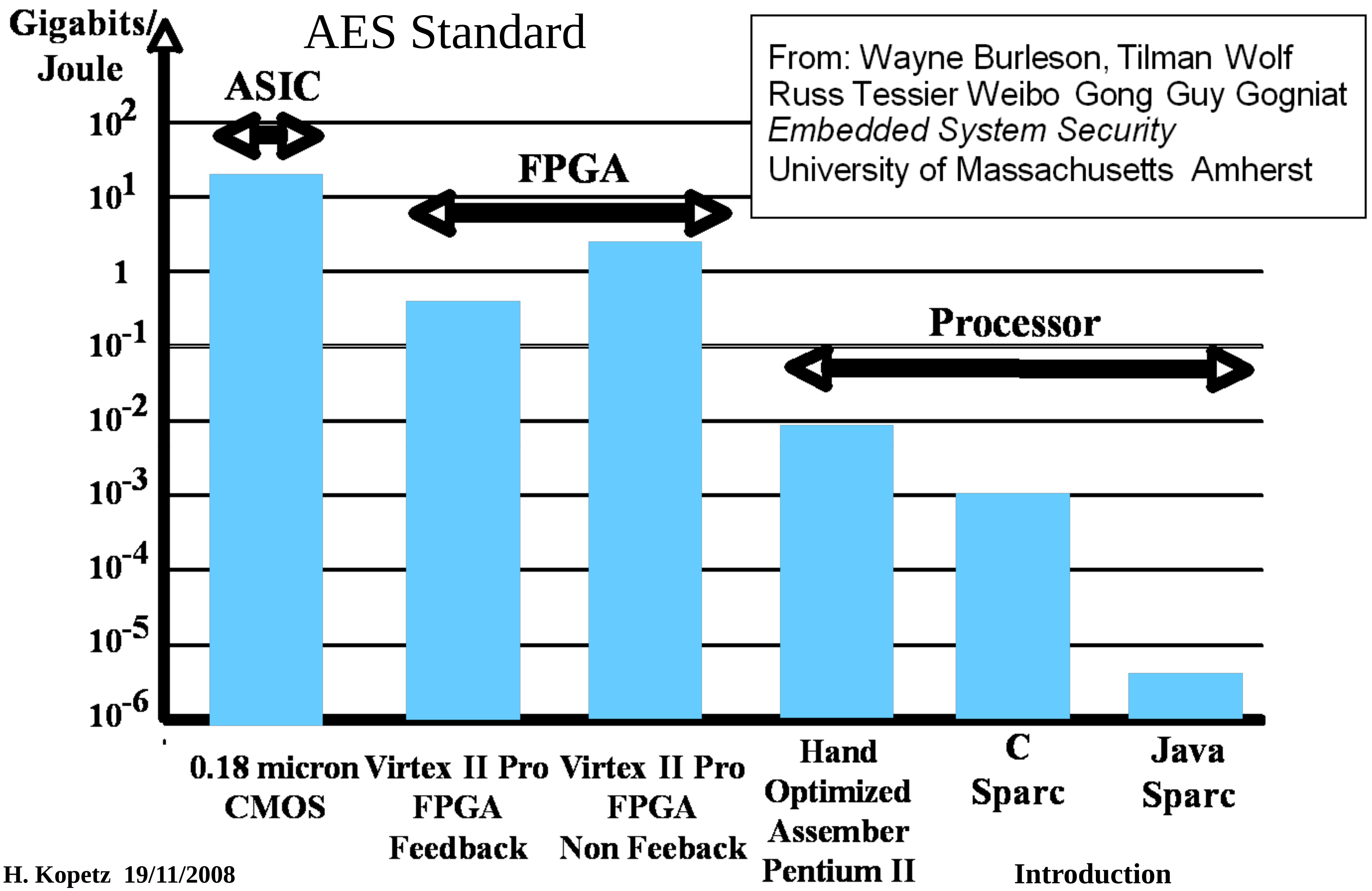


Performance Trends--Power



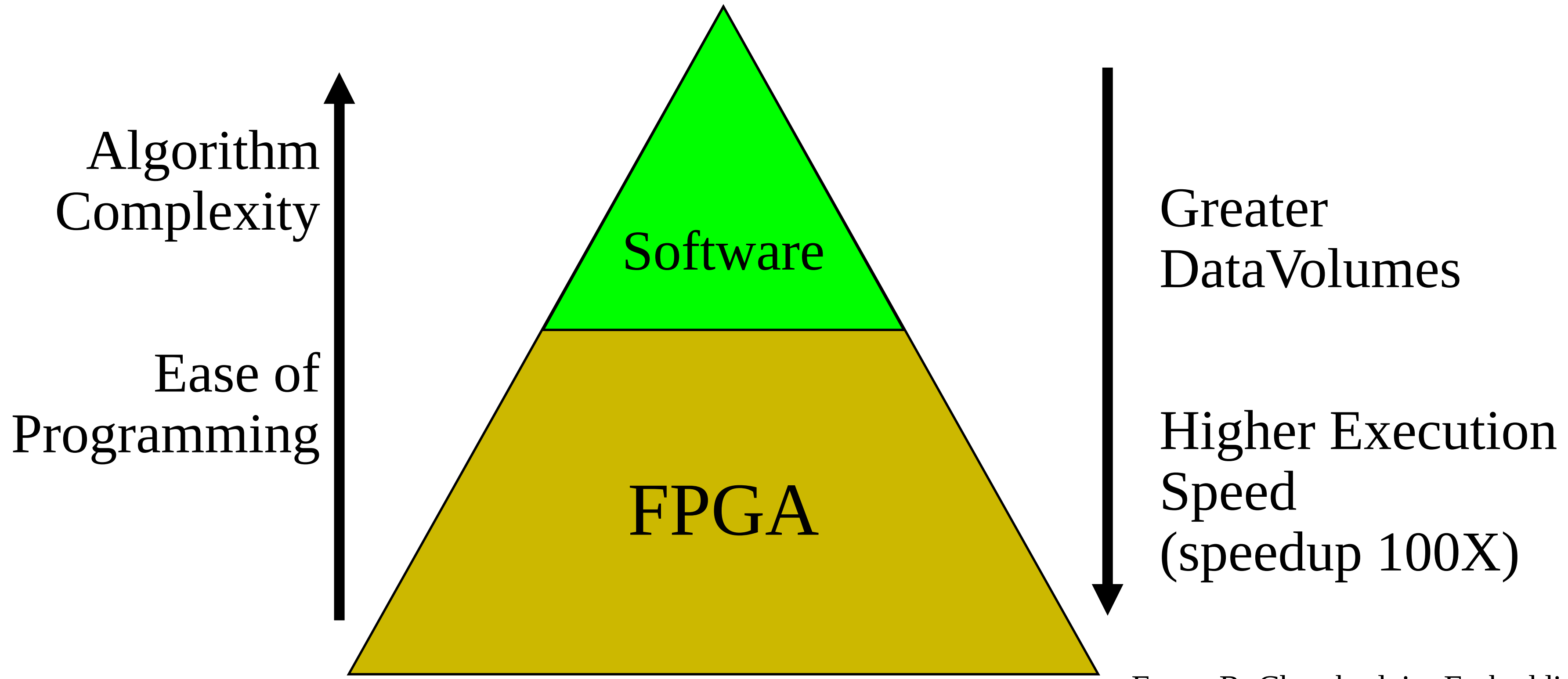
Ref: Lauwereins, Imec, MSOP 2006

Energy Efficiency of Different Implementations



From: Wayne Burleson, Tilman Wolf
Russ Tessier Weibo Gong Guy Gogniat
Embedded System Security
University of Massachusetts Amherst

Implementation Choices



From: R. Chamberlain, Embedding Applications within a Storage Aplliance Proc. HPEC 2005, p. 2

Example: Look for keywords in a set of documents:

(A *and* B) *or* (C *and* D)

Search for the occurrence of A,B,C,D in FPGA, connect the results in software