Modern embedded electronic systems design: a bird’s-eye view

Lars Svensson
lars.svensson@chalmers.se

Modern?

- Challenges:
  - Complexity
  - Performance
  - Power dissipation
  - Development rate
  - Manufacturing cost
  - …

Complexity

Complexity

Complexity

- 1997
- SMS
- Alarm
- Individual rings

Complexity

- 1997
- SMS
- Alarm
- Individual rings

Complexity

- 1997
- SMS
- Alarm
- Individual rings

- 2013
- Web browser
- Bluetooth
- WiFi
- Video
- Cameras
- Calendar
- MP3 player
- Android / Linux
- Color touch screen
- Multitouch

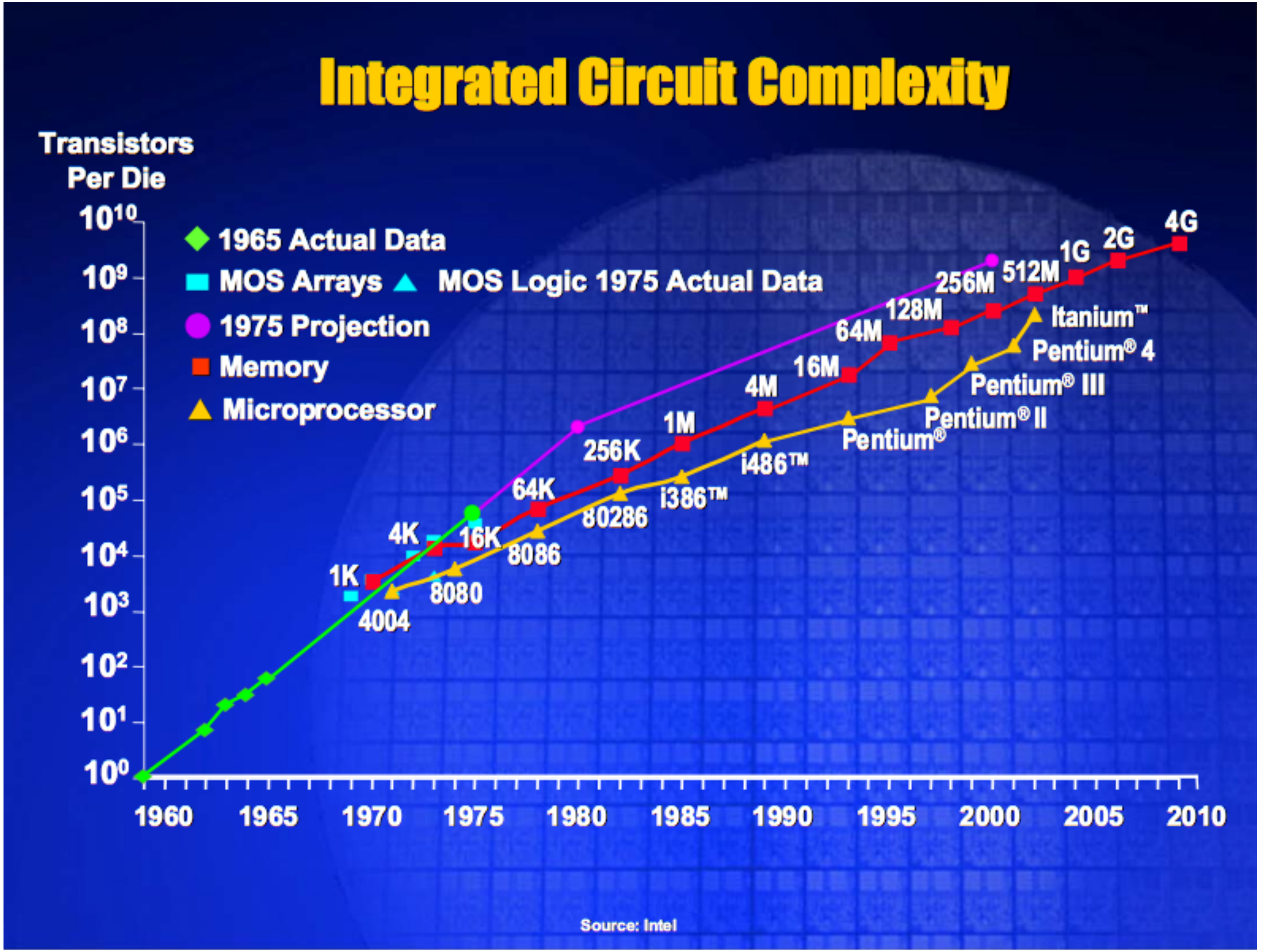
Complexity

- 1997
- SMS
- Alarm
- Individual rings

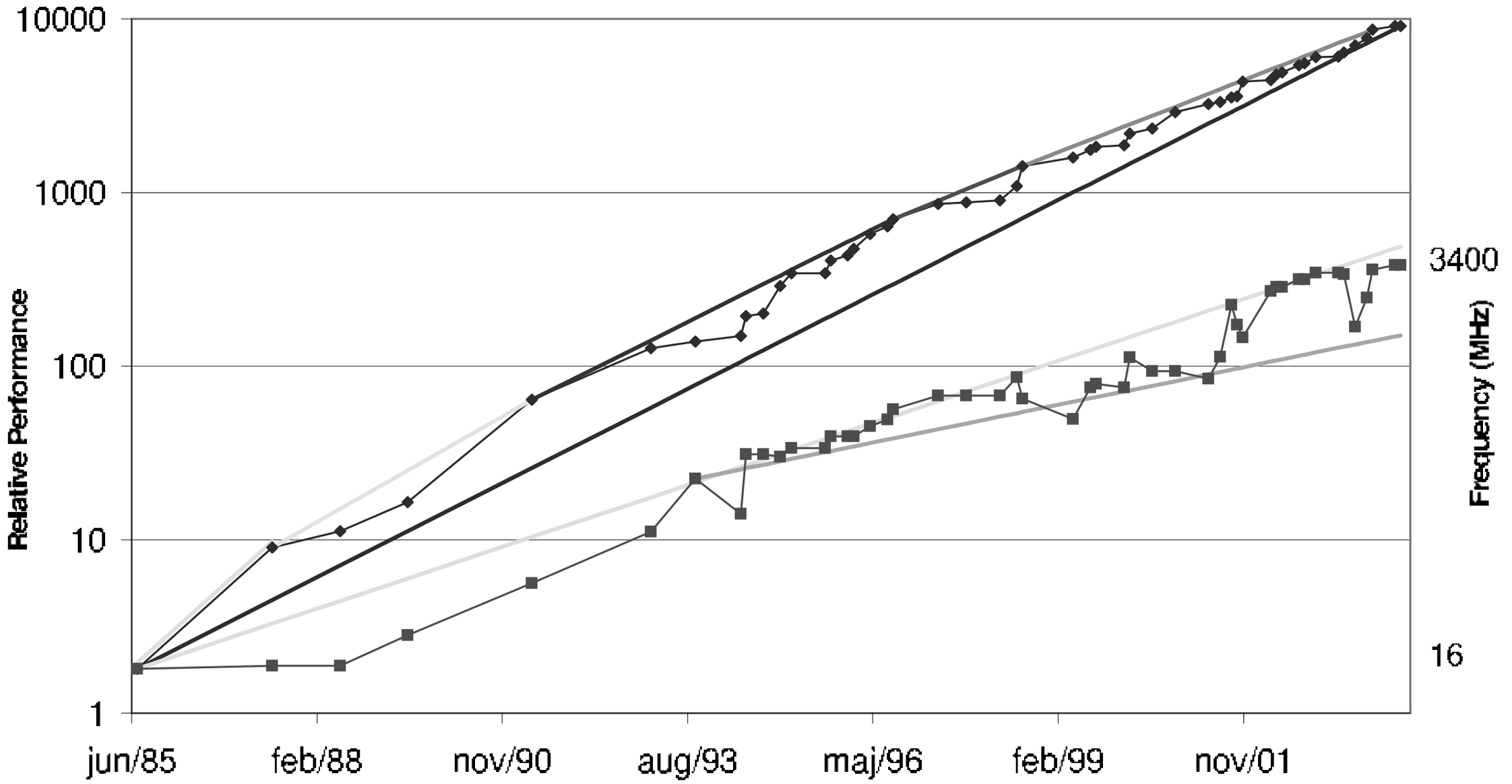
- 2013
- Web browser
- Bluetooth
- WiFi
- Video
- Cameras
- Calendar
- MP3 player
- Android / Linux
- Color touch screen
- Multitouch

- Address book
- Continuous internet connection
- Social app support
- App store …

Integrated Circuit Complexity

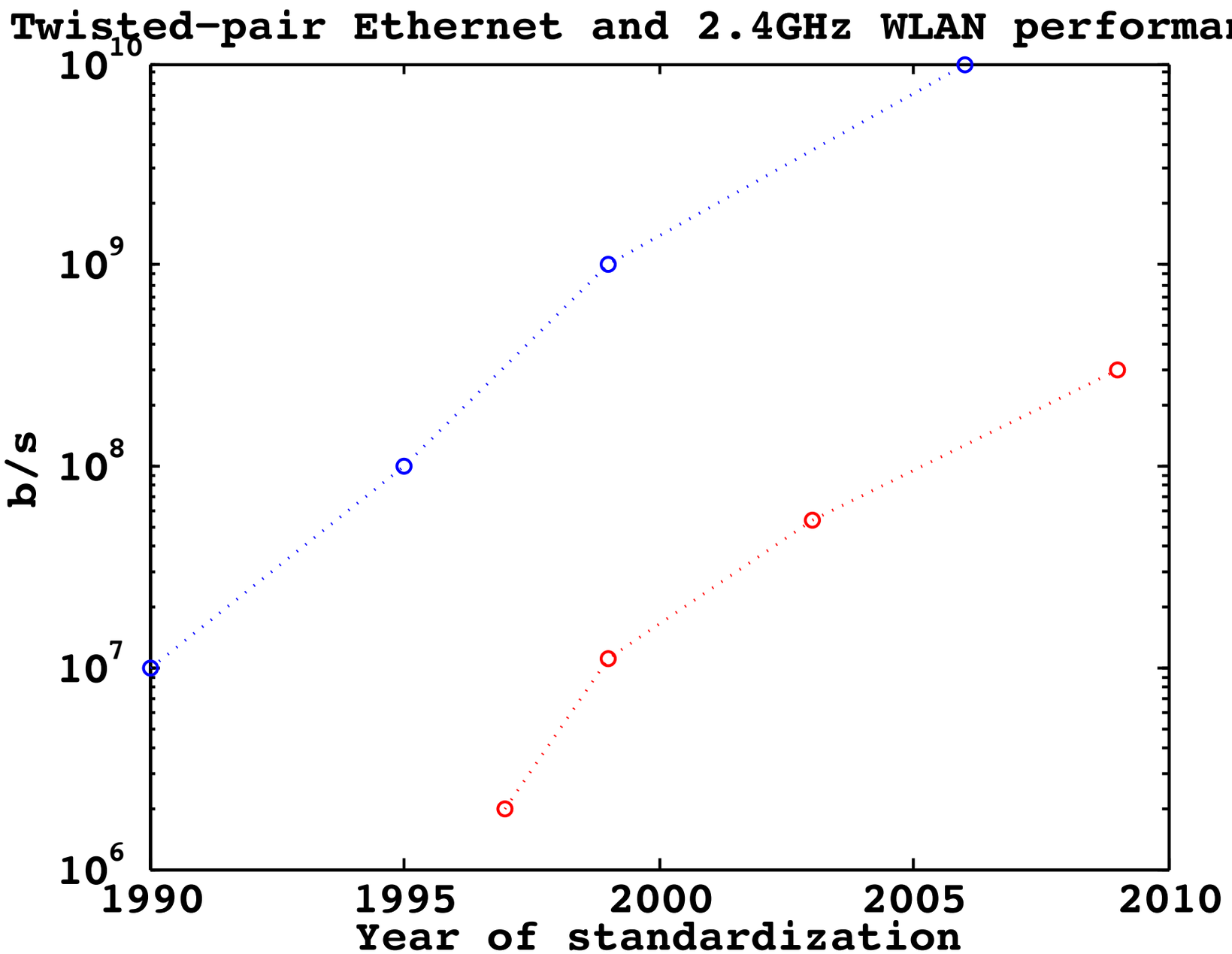


Performance

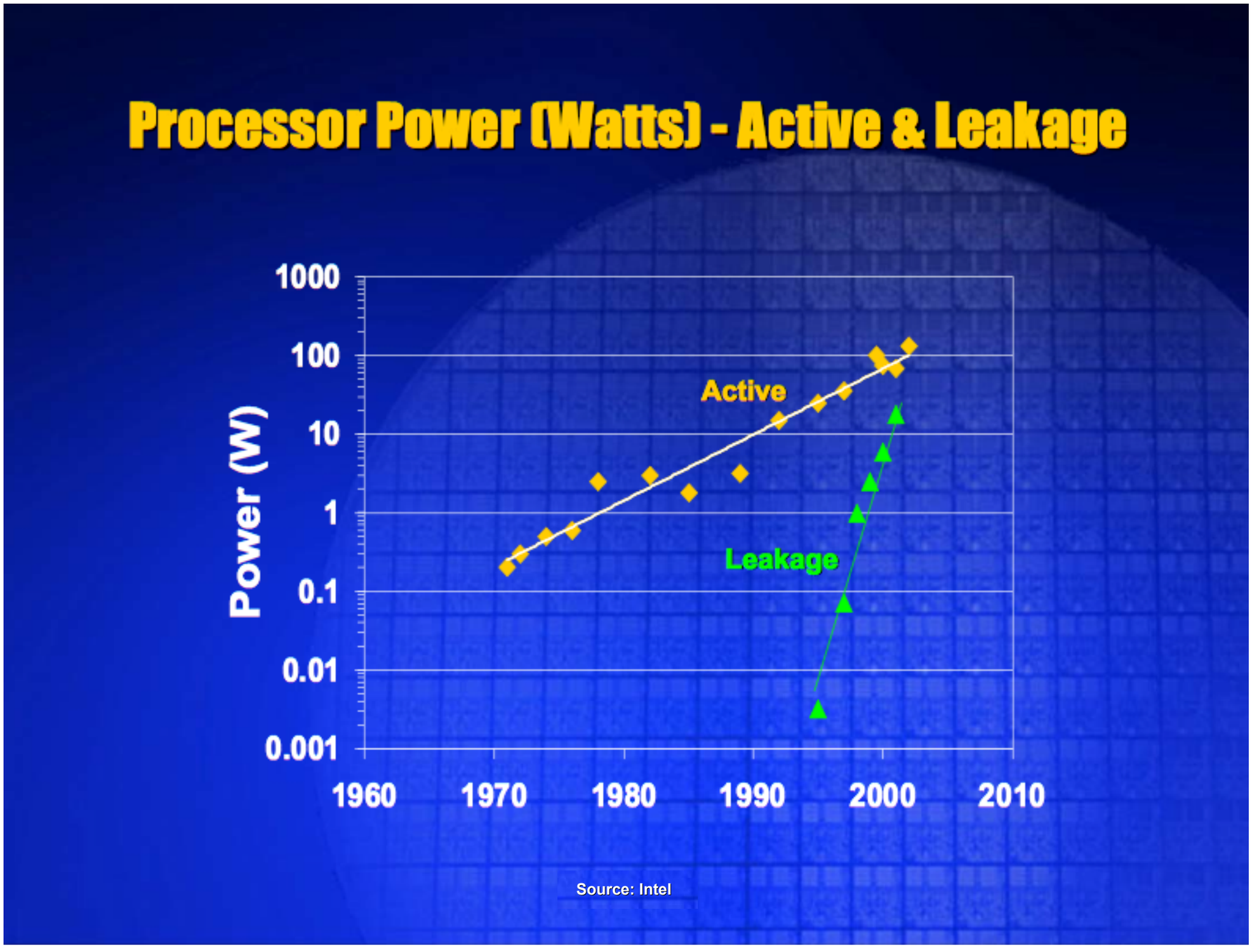


[Ekman, Warg, Nilsson 2004]

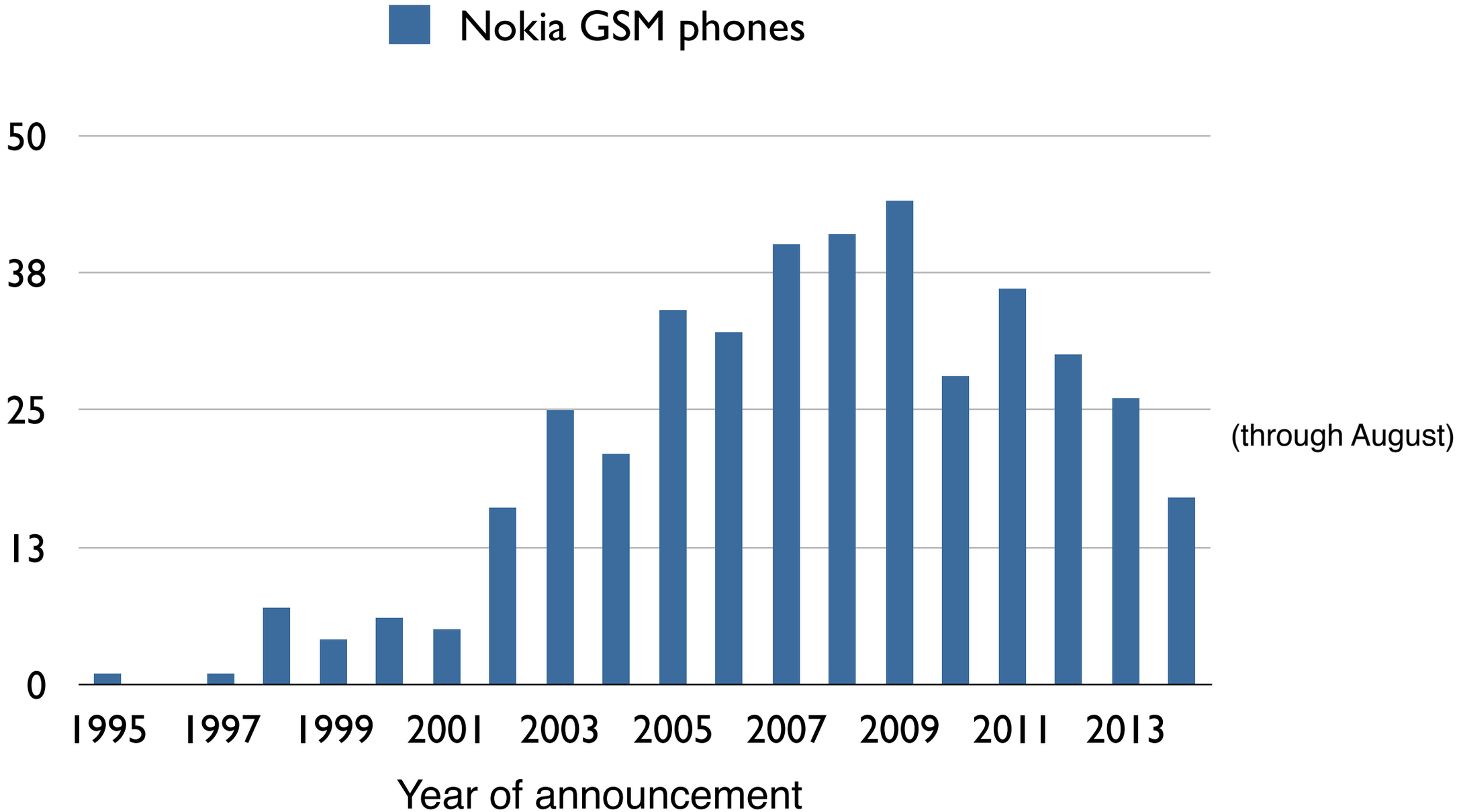
Performance



Power dissipation



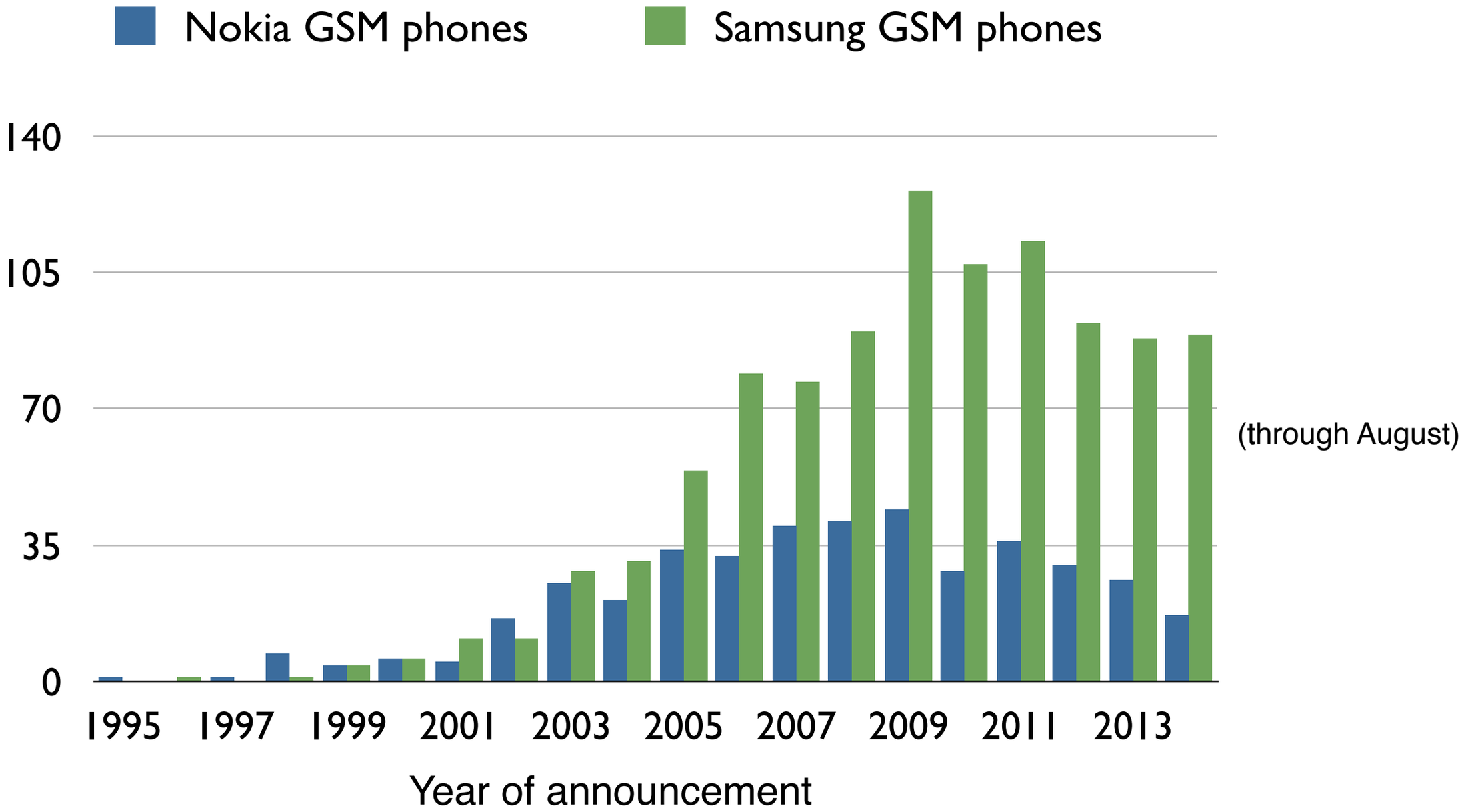
Development rate



(through August)

www.gsmarena.com

Development rate



(through August)

www.gsmarena.com

Typical class project (hw / sw)

- Scope: small
- Team: small (often size 1)
- Task: well-defined from start
- External dependencies: none
- Planning: simple
- Progress: linear
- Tools: well-known

Typical class project (hw / sw)

Typical real-world project (hw + sw + …)

- Scope: small
- Team: small (often size 1)
- Task: well-defined from start
- External dependencies: none
- Planning: simple
- Progress: linear
- Tools: well-known

Typical class project (hw / sw)

Typical real-world project (hw + sw + …)

- Scope: small
- Team: small (often size 1)
- Task: well-defined from start
- External dependencies: none
- Planning: simple
- Progress: linear
- Tools: well-known

Large and growing

Typical class project (hw / sw)

Typical real-world project (hw + sw + …)

- Scope: small
- Team: small (often size 1)
- Task: well-defined from start
- External dependencies: none
- Planning: simple
- Progress: linear
- Tools: well-known

Large and growing

“Big picture” very big

Typical class project (hw / sw)

Typical real-world project (hw + sw + …)

- Scope: small — Large and growing
- Team: small (often size 1) — “Big picture” very big
- Task: well-defined from start — Evolving rapidly
- External dependencies: none
- Planning: simple
- Progress: linear
- Tools: well-known

Typical class project (hw / sw)

Typical real-world project (hw + sw + …)

- Scope: small — Large and growing
- Team: small (often size 1) — “Big picture” very big
- Task: well-defined from start — Evolving rapidly
- External dependencies: none — Many critical ones
- Planning: simple
- Progress: linear
- Tools: well-known

Typical class project (hw / sw)

Typical real-world project (hw + sw + …)

- Scope: small — Large and growing
- Team: small (often size 1) — “Big picture” very big
- Task: well-defined from start — Evolving rapidly
- External dependencies: none — Many critical ones
- Planning: simple — Complex
- Progress: linear
- Tools: well-known

Typical class project (hw / sw)

Typical real-world project (hw + sw + …)

• Scope: small	Large and growing
• Team: small (often size 1)	“Big picture” very big
• Task: well-defined from start	Evolving rapidly
• External dependencies: none	Many critical ones
• Planning: simple	Complex
• Progress: linear	Chaotic
• Tools: well-known	

Typical class project (hw / sw)

Typical real-world project (hw + sw + …)

• Scope: small	Large and growing
• Team: small (often size 1)	“Big picture” very big
• Task: well-defined from start	Evolving rapidly
• External dependencies: none	Many critical ones
• Planning: simple	Complex
• Progress: linear	Chaotic
• Tools: well-known	Hmm… wonder what this does?

Big Issue

- "Everything" gets more difficult, often exponentially with time
- How handle this explosion?
- How to design and develop, today?

Q: How to design and develop, today?

- (Required) A: a design/development method, process, methodology, or approach, which handles…
  - …large and larger projects
  - …full range of development tasks
  - …many kinds of requirements
  - …activity scheduling, cost estimates, risk mitigation, etc
- Concept examples: Waterfall, Agile, Lean

Q: How to design and develop, today?

- (Required) A: a design/development method, process, methodology, or approach, which handles…
  - …large and larger projects
  - …full range of development tasks
  - …many kinds of requirements
  - …activity scheduling, cost estimates, risk mitigation, etc

- Concept examples: Waterfall, Agile, Lean

1. Waterfall model

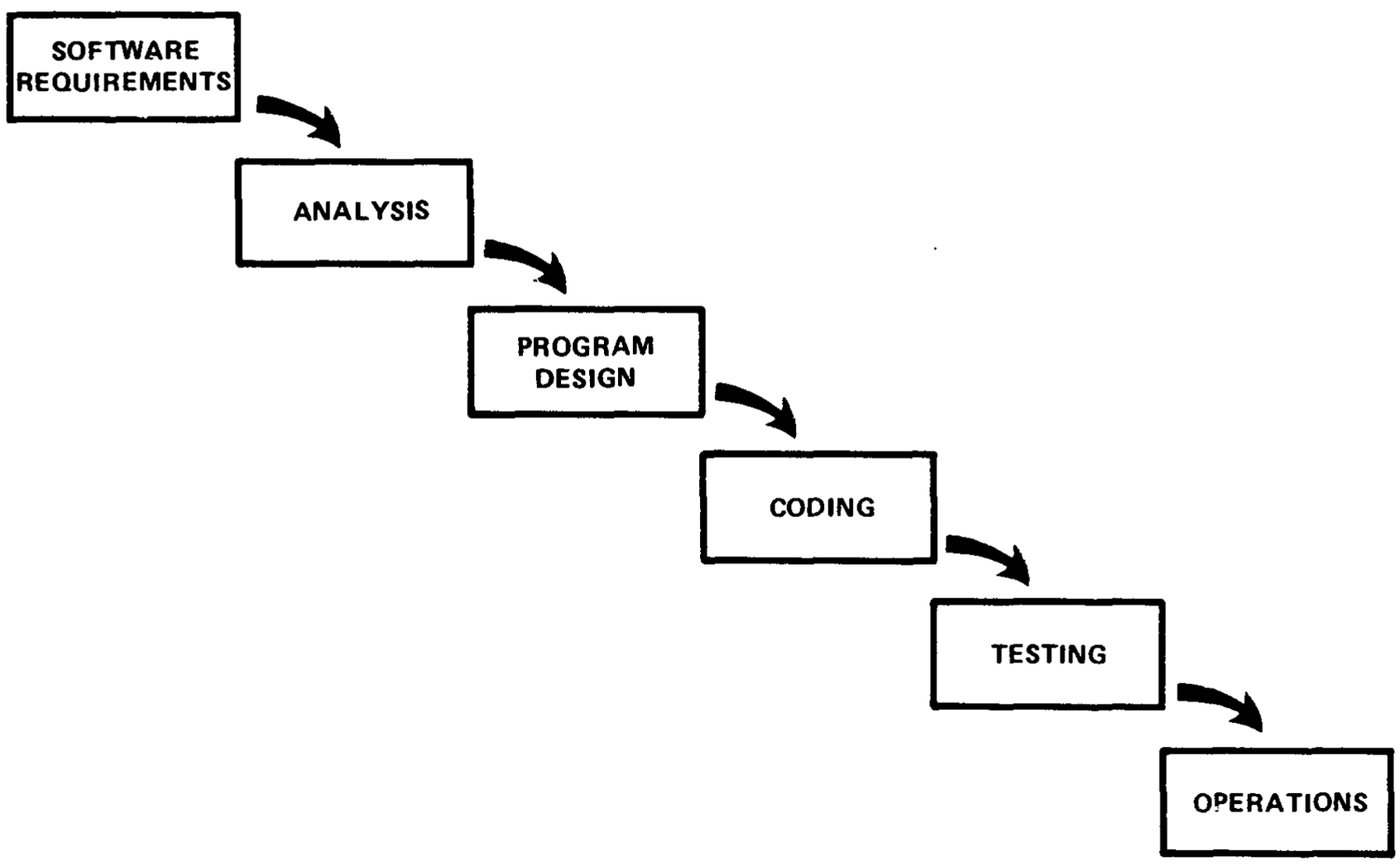
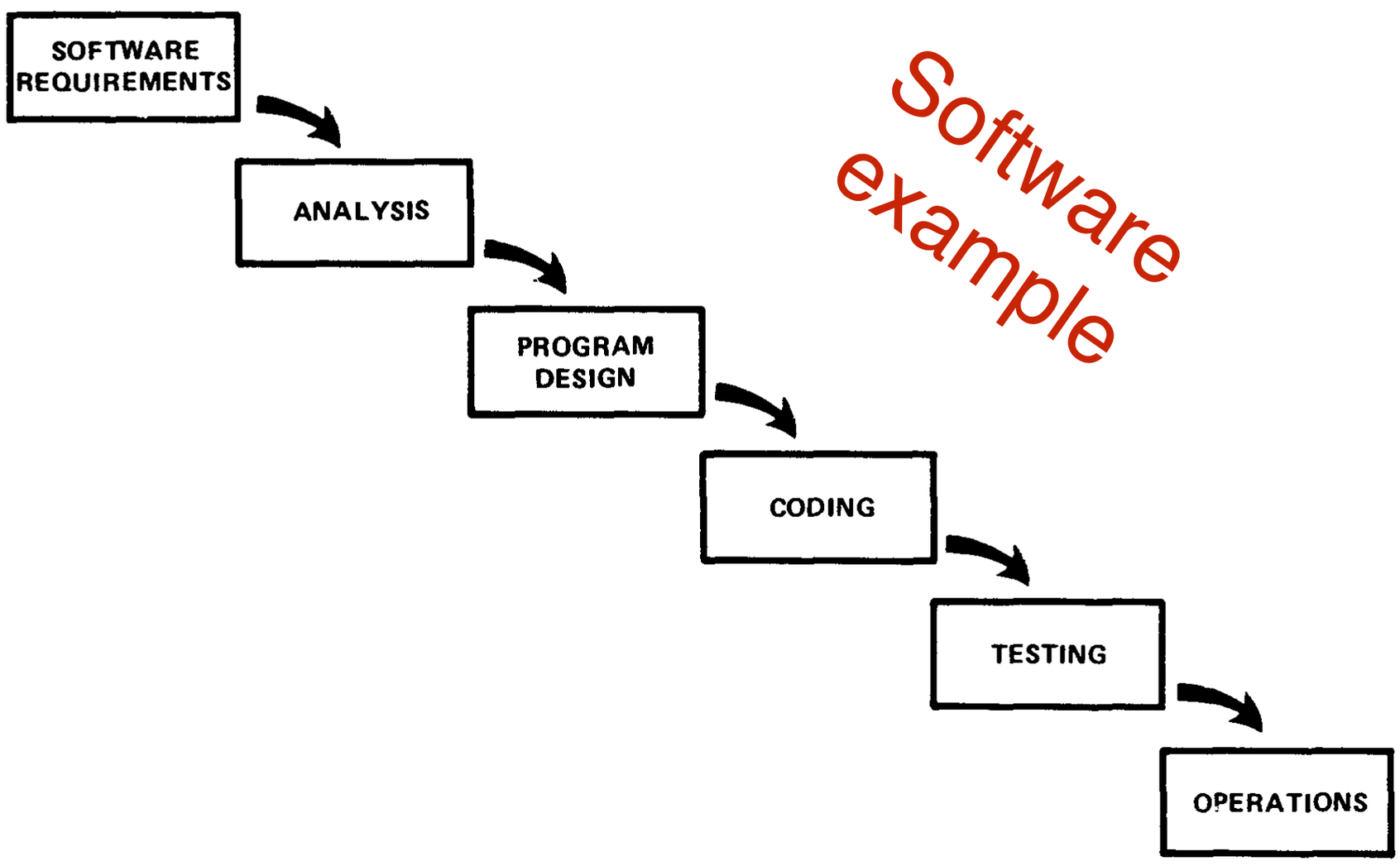


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

[Royce 1970]

1. Waterfall model



Software example

Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

[Royce 1970]

Waterfall characteristics

- Proceed in orderly fashion from step to step
- Finish each task before starting the next one (never backtrack)
- Scrupulously document everything
- Pure Waterfall unrealistic even when first published…

Waterfall characteristics

- Proceed in orderly fashion from step to step
- Finish each task before starting the next one (never backtrack)
- Scrupulously document everything
- Pure Waterfall unrealistic even when first published…

Why?

Selected Waterfall problems

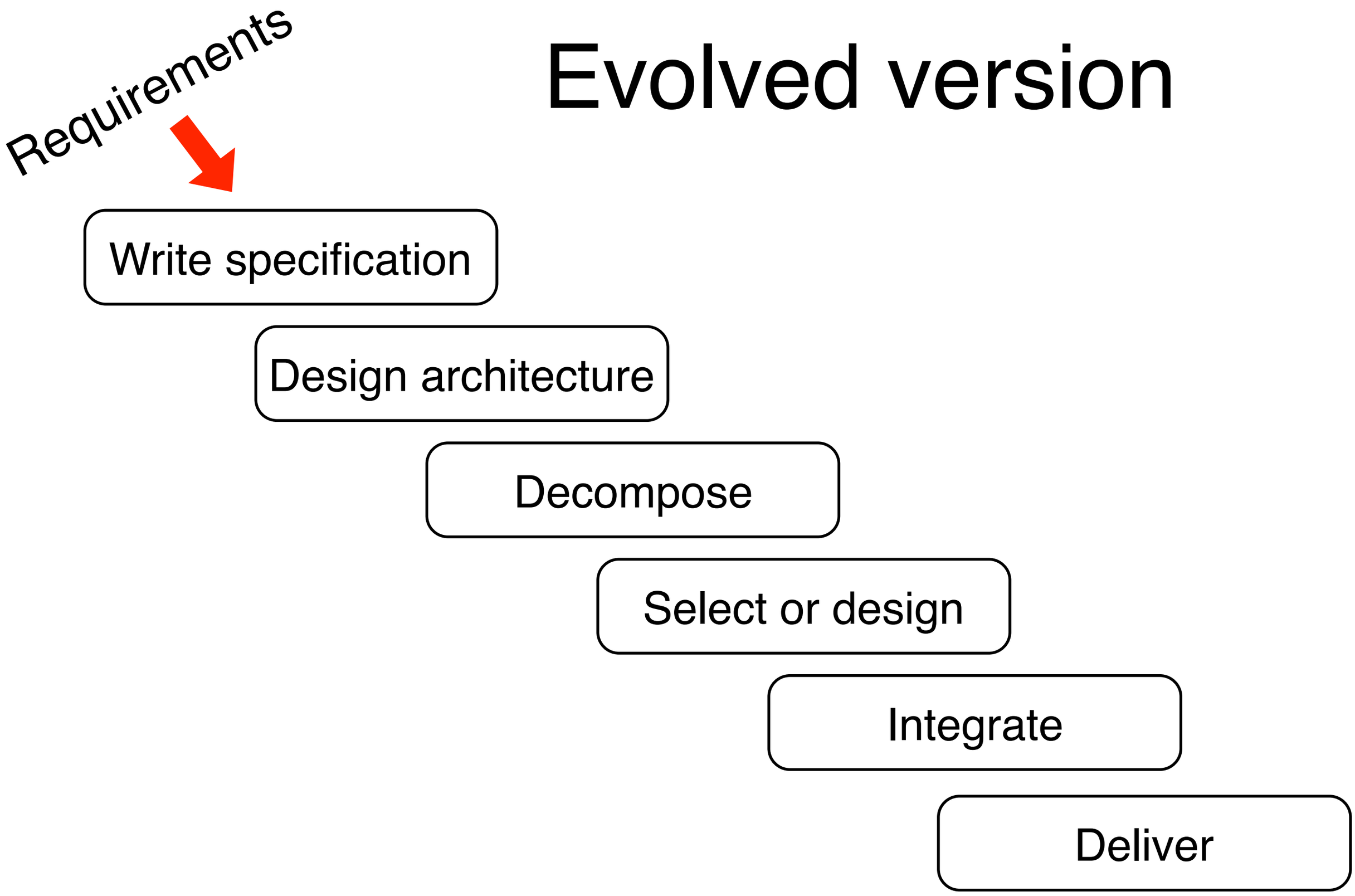
- Monolithic; no provisions for design re-use
  - Very few projects start with clean sheet
- Assumes static environment
  - Late spec changes are not handled
- “Clean” hand-offs are rare
  - Iterations necessary in practice
- Late-stage resources idle at start

Selected Waterfall problems

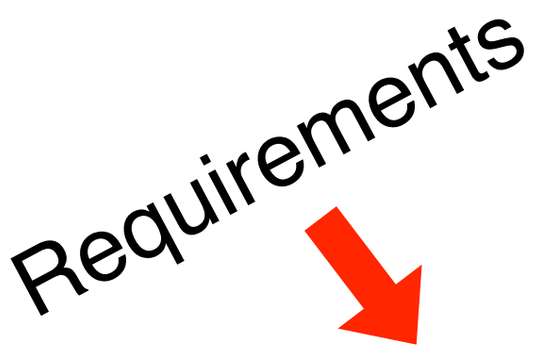
- Monolithic; no provisions for design re-use
  - Very few projects start with clean sheet
- Assumes static environment
  - Late spec changes are not handled
- “Clean” hand-offs are rare
  - Iterations necessary in practice
- Late-stage resources idle at start

DOES NOT SCALE

Evolved version



Evolved version



Write specification

Full detailed specification!

Design architecture

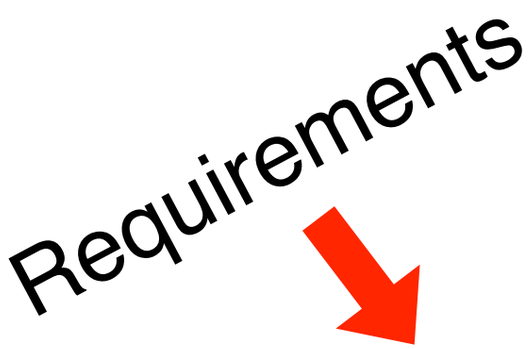
Decompose

Select or design

Integrate

Deliver

Evolved version



Write specification

Full detailed specification!

Design architecture

Hardware + software

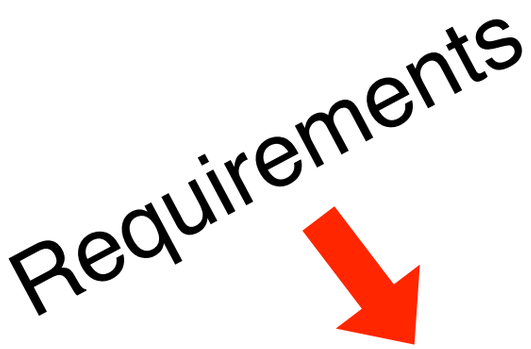
Decompose

Select or design

Integrate

Deliver

Evolved version



Write specification — Full detailed specification!

Design architecture — Hardware + software

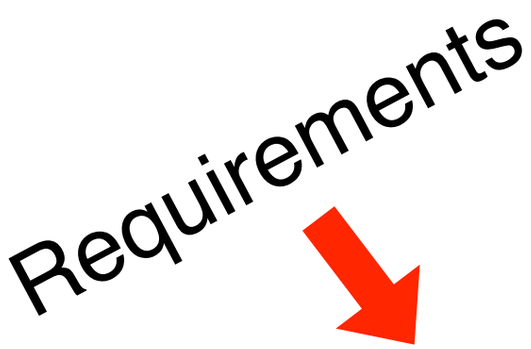
Decompose — Distribute requirements

Select or design

Integrate

Deliver

Evolved version



Write specification — Full detailed specification!

Design architecture — Hardware + software

Decompose — Distribute requirements

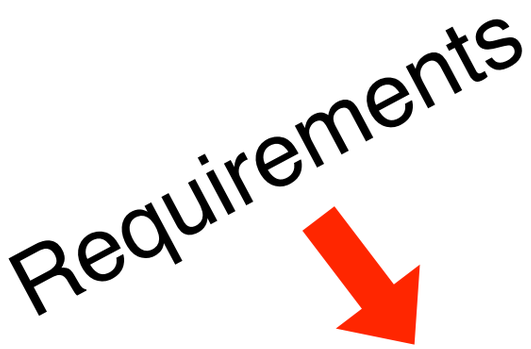
"Design" means *recurse*

Select or design

Integrate

Deliver

Evolved version



Write specification — Full detailed specification!

Design architecture — Hardware + software

Decompose — Distribute requirements

"Design" means *recurse*

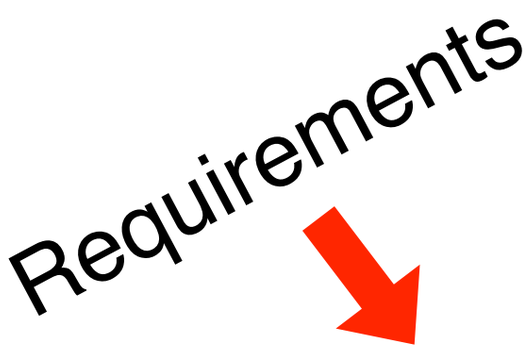
Verify that reqs are met

Select or design

Integrate

Deliver

Evolved version



Write specification — Full detailed specification!

Design architecture — Hardware + software

Decompose — Distribute requirements

"Design" means *recurse*

Select or design

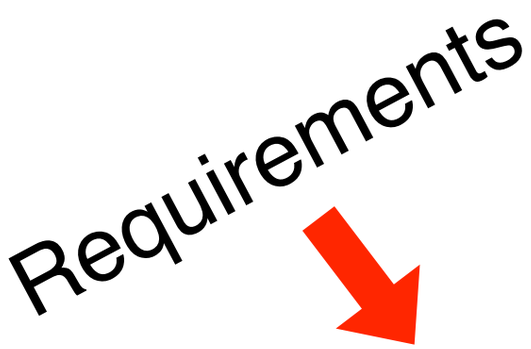
Verify that reqs are met

Integrate

$$:-)

Deliver

Evolved version



Write specification — Full detailed specification!

Design architecture — Hardware + software

Decompose — Distribute requirements

"Design" means *recurse*

Select or design

Verify that reqs are met

Integrate

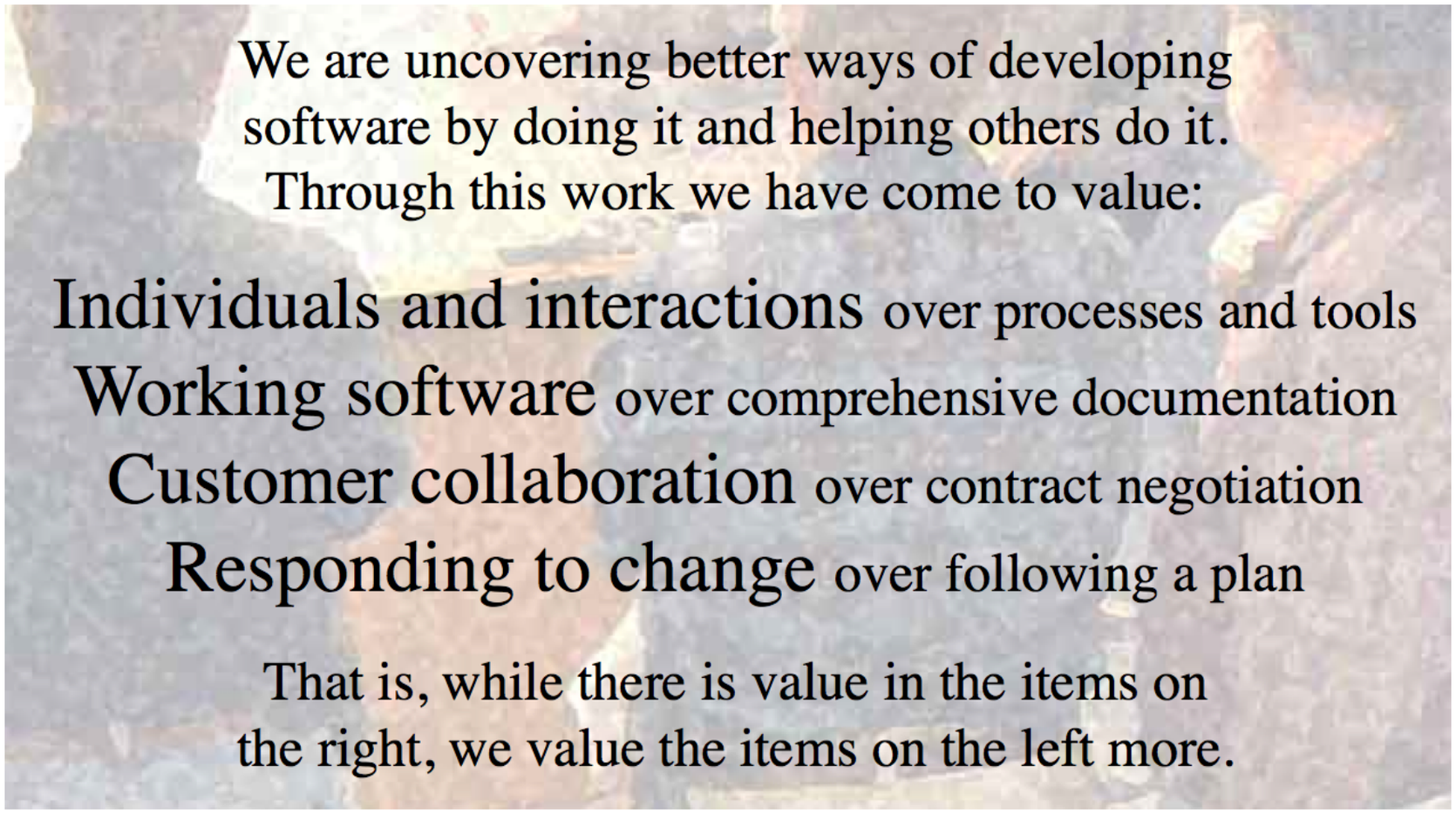
Remaining issues?

$$:-)

Deliver

2. Agile methods

- Less of a blueprint, more of a philosophy
  - Principles, practices, tools embody the philosophy
- Manifesto:



 [www.agilemanifesto.org]

Agile practices (examples)

- User stories define final product
- Test-driven development relies on automatic application of functionality tests
- Repeated refactoring for incremental design during implementation
- Pair programming reduces risk for “smart” solutions which work for only 99% of the cases
- Release early + often to get feedback

Agile practices (examples)

- User stories define final product
- Test-driven development relies on automatic application of functionality tests
- Repeated refactoring for incremental design during implementation
- Pair programming reduces risk for “smart” solutions which work for only 99% of the cases
- Release early + often to get feedback

All applicable to hardware/system design

Agile practices (examples)

- User stories define final product
- Test-driven development relies on automatic application of functionality tests
- Repeated refactoring for incremental design during implementation
- Pair programming reduces risk for “smart” solutions which work for only 99% of the cases
- Release early + often to get feedback

All applicable to hardware/system design

3. Lean development

- Origin in manufacturing rather than in software development
  - "Lean manufacturing" @ Toyota, 1980's
  - Introduced in software development ~2000 (Poppendieck & Poppendieck)
- Key ideas:
  - Optimize the whole
  - Eliminate waste
  - Build quality in
  - Learn constantly
  - Deliver fast
  - Engage everyone
  - Keep getting better

[Poppendieck & Cusumano, 2012]

3. Lean development

- Origin in manufacturing rather than in software development
  - "Lean manufacturing" @ Toyota, 1980's
  - Introduced in software development ~2000 (Poppendieck & Poppendieck)
- Key ideas:
  - Optimize the whole
  - Eliminate waste
  - Build quality in
  - Learn constantly
  - Deliver fast
  - Engage everyone
  - Keep getting better

Similarities w/ Agile?

[Poppendieck & Cusumano, 2012]

3. Lean development

- Origin in manufacturing rather than in software development
  - "Lean manufacturing" @ Toyota, 1980's
  - Introduced in software development ~2000 (Poppendieck & Poppendieck)
- Key ideas:
  - Optimize the whole
  - Eliminate waste
  - Build quality in
  - Learn constantly
  - Deliver fast
  - Engage everyone
  - Keep getting better

Similarities w/ Agile?

Differences wrt Waterfall?

[Poppendieck & Cusumano, 2012]

Software vs. system development

- Here, “system” == software + digital hardware + analog circuits + packaging + services + …
- Software development methods cannot be applied blindly!
- Principles of Agile and Lean inspire engineering development in all fields today

Pitfalls (for <u>any</u> approach)

- Focus on tools or practices, w/o understanding <u>why</u> they are important
- Missing the big picture
- Lack of skills and knowledge of specific topic area

It is really all about learning.

- Designers / developers need knowledge and skills
  - You'll learn some in University courses
- Design and development <u>is</u> learning

Skills / knowledge: aspects of design

1. Specifications

- Many kinds:
  - Documents in natural language (English, etc)
  - Executable specifications
  - Formal specifications

- Must include:
  - functionality/behavior
  - interfaces (to hardware and software)

- May include:
  - timing
  - performance
  - power
  - testability …

LABS

DAT096

1. Specifications

- Many kinds:
  - Documents in natural language (English, etc)
  - Executable specifications
  - Formal specifications

What makes each kind useful?

- Must include:
  - functionality/behavior
  - interfaces (to hardware and software)
- May include:
  - timing
  - performance
  - power
  - testability …

LABS

DAT096

2. System architecture

- Major design decisions
  - Hardware vs. software **DAT096**
  - Analog vs. digital **DAT116**
  - Mechanical construction **MKM105**
  - Processor/memory hierarchies **EDA332 DAT105**
  - … **EDA283**

3. Hard vs Soft

- Hardware: inflexible, high performance capability
- Software: flexible (well…), limited performance
- Reconfigurable hardware: flexible, intermediate performance, high cost per part
- More in later lecture

4. Analog vs Digital

- Digital embedded electronic systems interact with analog world
- Where to draw the A/D border?
- Often major influence on performance, cost

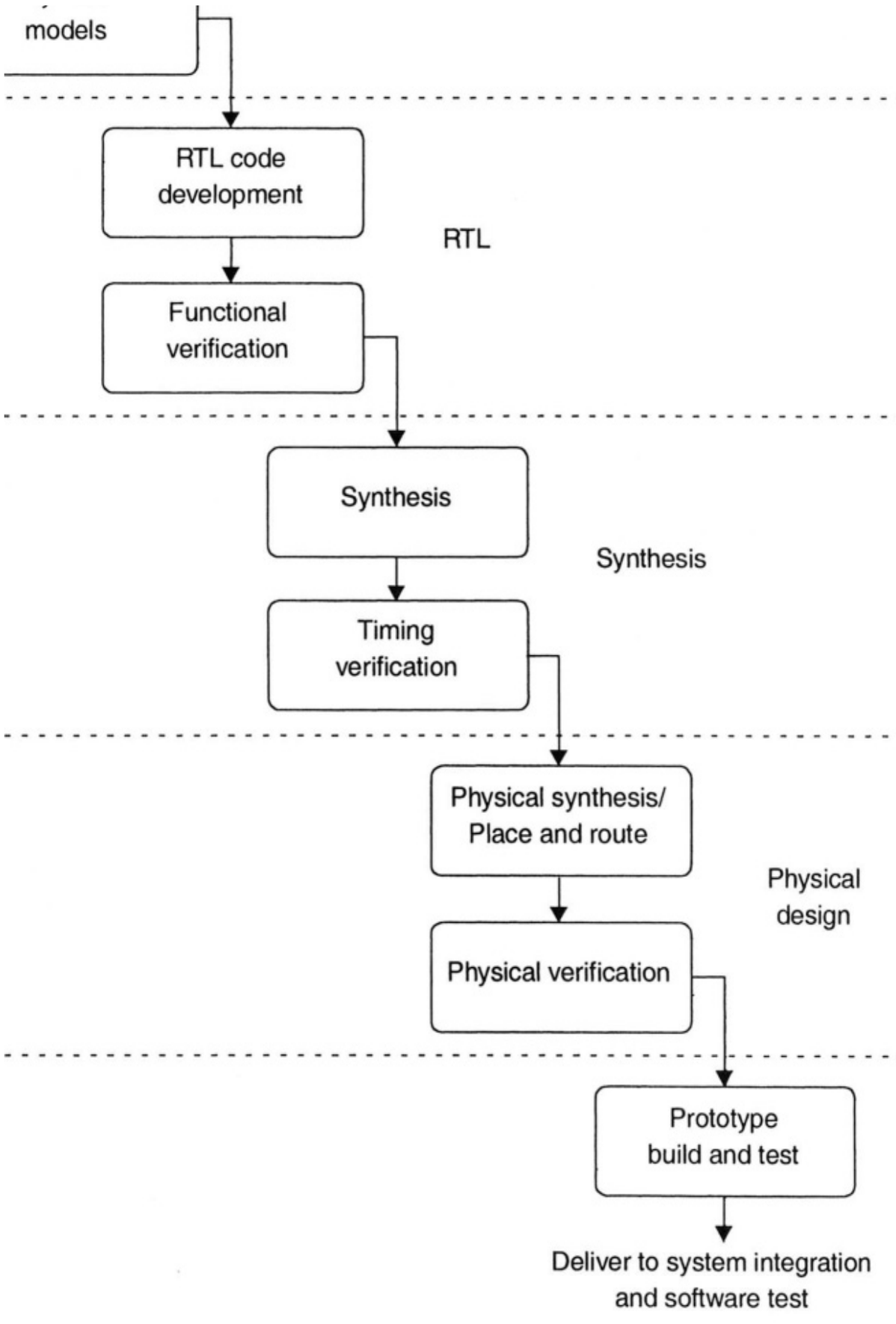
DAT116

5. Digital design (or selection)

- Design what's not available off-the-shelf
  - Hardware Description Languages **LABS**
- Design methods rely heavily on CAD tools **DAT110**
- Still necessary to understand the underlying circuit behavior **MCC092**
  - Determines achievable performance

Digital design flow

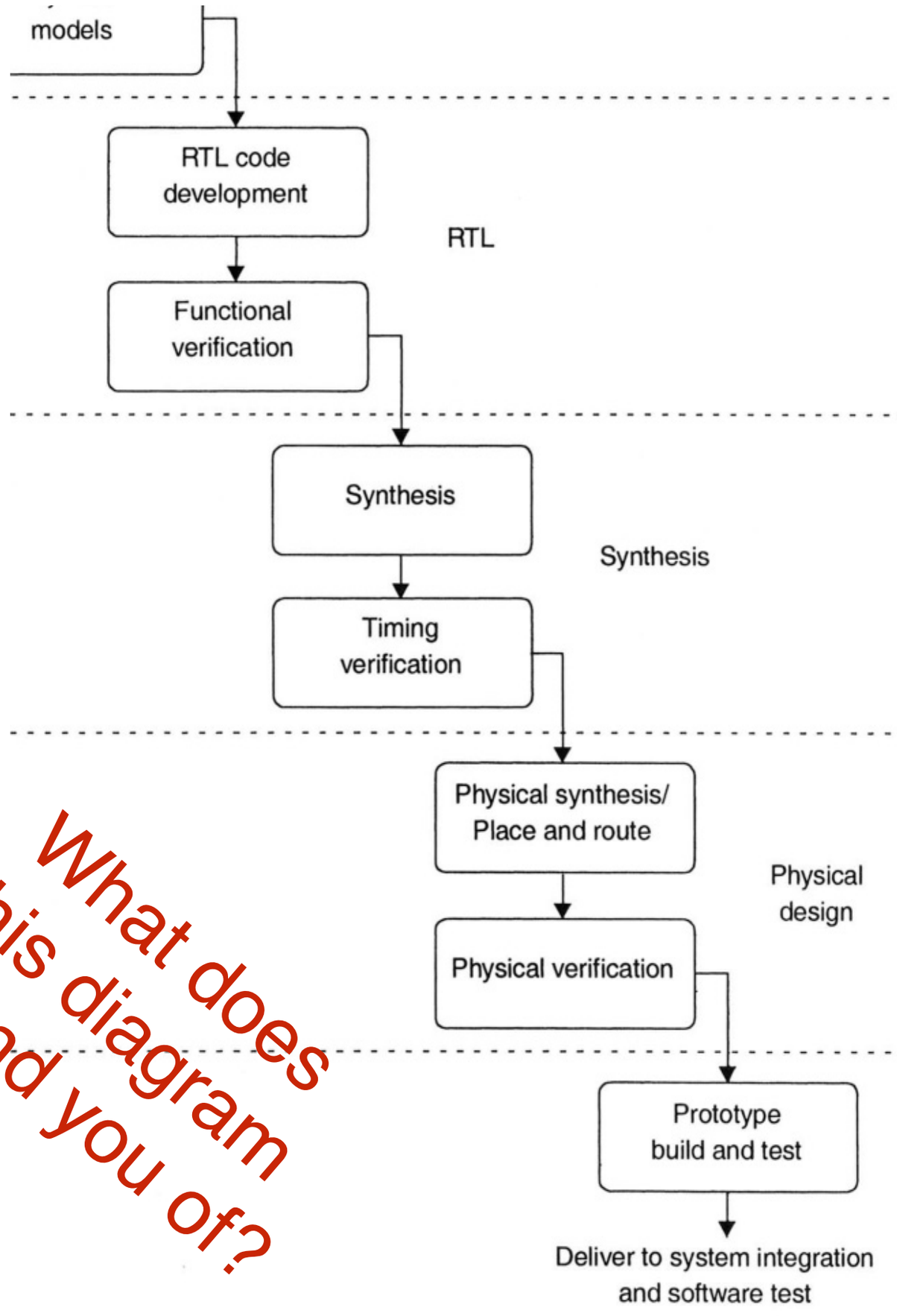
- Stages as in this diagram
  - Additions, deletions common
- Iterations typically necessary



[Keating, Bricard 2002.]

Digital design flow

- Stages as in this diagram
  - Additions, deletions common
- Iterations typically necessary



What does this diagram remind you of?

[Keating, Bricard 2002.]

6. Integration & verification

- Putting the pieces together
- Very time-consuming process!
  - Must be considered from the outset!
- Errors must be patched
  - Software vs. hardware

LABS

DAT096

Summary

- Electronic-system design complexity ever-growing
  - Includes software, analog/digital, packaging, batteries, …
- Design process is much more than putting electronic components together!
  - Designer needs “vertical” insight
- Career typically starts at implementation level, moves “up”

This week

- Two more lectures by Sven Knutsson
  - VHDL + the lab series
  - Note: room EF, EE (two floors up)
- VHDL kick-start workshop
  - Wednesday (+ Friday)
- Readings on design approaches (final slide)

To do:

- Get registered for the course!
- Verify access to PingPong web system!
- Fill out the pre-workshop survey!
  - In Ping-Pong, under “Contents”
- Watch VHDL movie (again)!
- Optionally complete your VHDL library!

Background reading

- Royce: Managing the development of large software systems (1970) [available in PingPong]
- Wikipedia-article on Agile Software Development
- Poppendieck, Cusumano: Lean Software Development: a tutorial (2012) [available in PingPong]