Science vs. Innovation in computer architecture research

Raphael 'kena' Poss
University of Amsterdam, the Netherlands

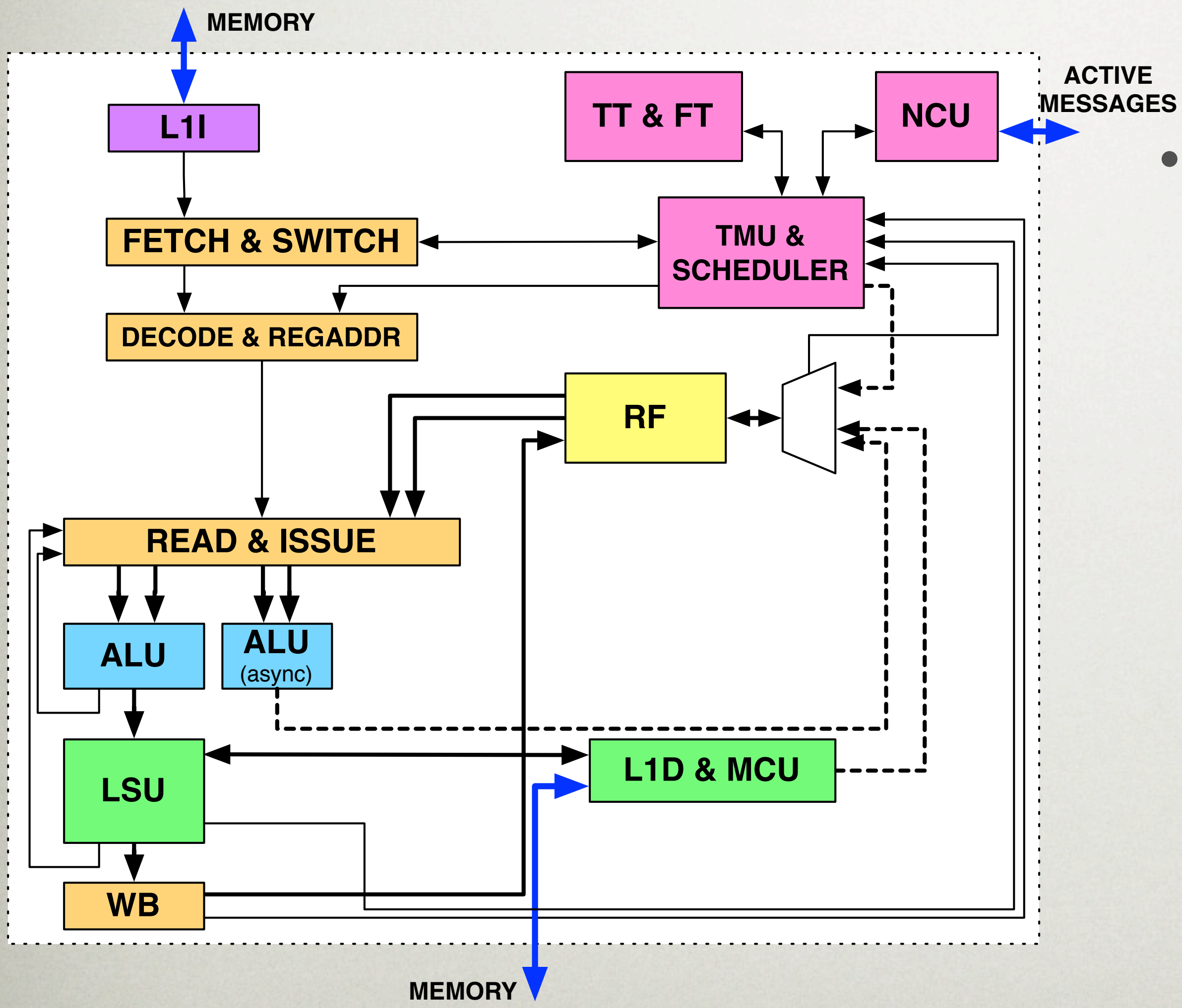
IvI Colloquium
May 31st, 2012

Recent thesis "On the realizability of hardware microthreading"

Current on-chip parallelism is based on legacy

- Historical focus on **single-thread performance**
  (developments in general-purpose processors: registers, branch prediction, prefetching, out-of-order execution, superscalar issue, trace caches, etc.)
- Legacy heavily **biased towards single threads**:
  - Symptom: **interrupts** are the **only way** to signal asynchronous external events
  - Retro-fitting **hardware multithreading** is **difficult** because of the sequential core's complexity
- **What if...**
  we **redesigned general-purpose processors, assuming concurrency is the norm in software**?

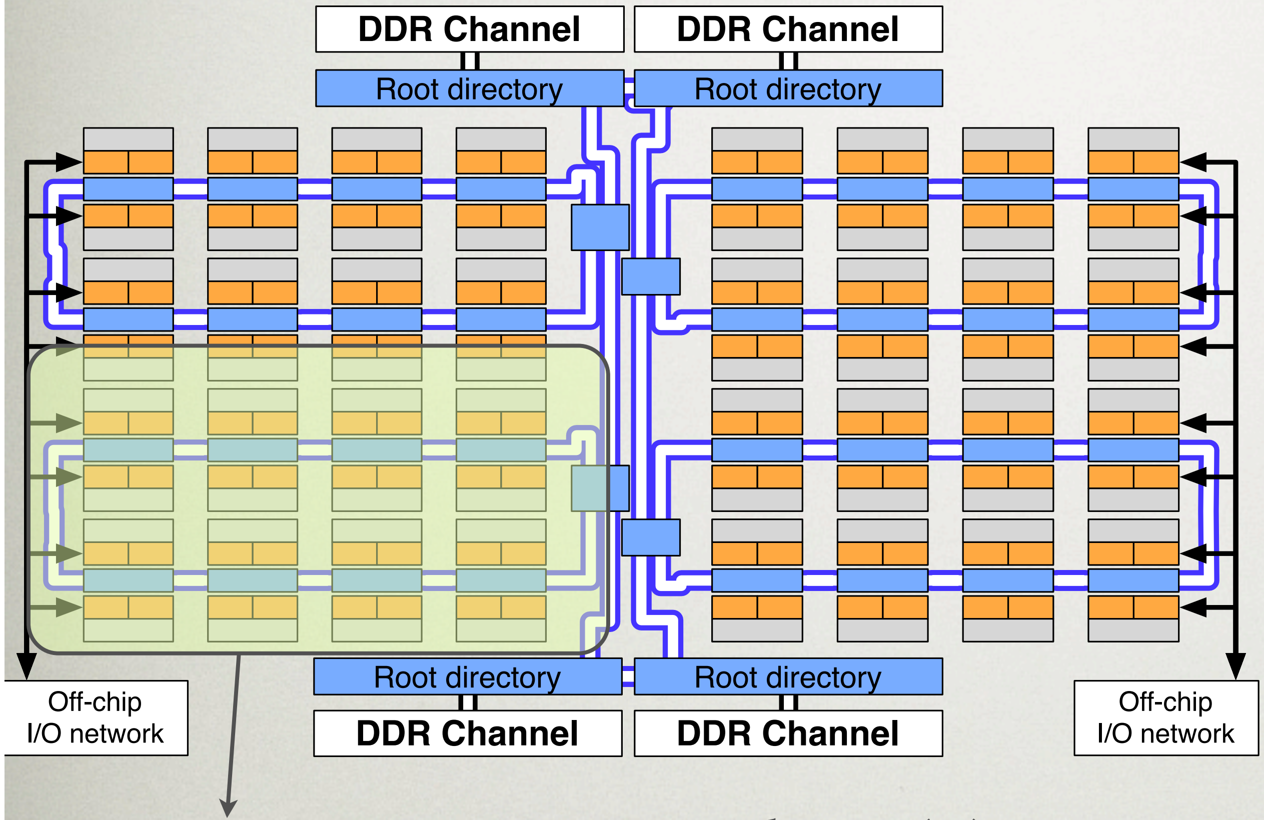
Microgrids of D-RISC cores



- D-RISC cores: **hardware multithreading** + **dynamic dataflow scheduling**
  - **fine-grained threads**: 0-cycle thread switching, <2 cycles creation overhead
  - **ISA instructions** for thread management
  - dedicated hardware processes for **bulk creation and synchronization**
  - **No preemption/interrupts;** events create new threads

In-order, single-issue RISC: small, cheaper, faster/watt

Example 128-core Microgrid



Approximate size of one Nehalem (i7) core for comparison

- **32000+ hw threads**
- 5MB **distributed cache**
- **shared MMU = single virtual address space,** protection using **capabilities**
- **Weak cache coherency**
- **no support for global memory atomics** – instead synchronization using remote register writes

Area estimates with CACTI: 100mm2 @ 35nm

A perspective shift

Core i7	**Function call** with 4 registers spilled **30-100 cycles**	**Predictable loop** requires branch predictor + cache prefetching to maximize utilization 1+ cycles per iteration overhead
D-RISC with TMU in hardware	**Bulk thread creation** of 1 thread, 31 “fresh” registers **~15 cycles** **(7c sync, ~8c async)**	**Thread family** 1 thread / “iteration” reuses common TMU and pipeline no BP nor prefetch needed no per-iteration overhead

Results, what's next?

- ✓ built **enough infrastructure to fit the F/OSS landscape**
  – yet can't reuse most existing OS code: ***no interrupts, no traps***
- ✓ as planned, **higher performance per area and per watt**
  – via hand-coded benchmarks: ***granularity in SPEC is too coarse***
- **Follow-up research areas:**
  - *Internal* issues: memory consistency, scalable cache protocols, ISA semantics, etc.
  - *External* issues from outside architecture: how to virtualize? how to map tasks over so many "workers"? how to port existing OS code?
  - *Fundamental* issues: concurrent complexity theory?

Preliminary outcome

- **We can make smarter processors but they look & feel different to system developers.**
  - Analogy: a new hexagonal Lego unit
- To gain traction: **demonstrate** the benefits in **applied problems**
- But this seems **hard** to all actors in our field, why is that?

Generative computer architecture

"Ideas vs. realization" - not!

- Common fallacy:
  "**coming up with an idea ≠ implementing this idea**"
  "Ideas are free, but execution is priceless" – Scott Ginsberg
- In computer architecture:
  - some people *specify* components
    - often using smaller components as sub-parts
  - other people *integrate* designs into systems
  - other people *deploy/validate* systems to applications
- **Computer architecture is an ecosystem...**
  different people, different responsibilities
- **... in symbiosis with software ecosystems**

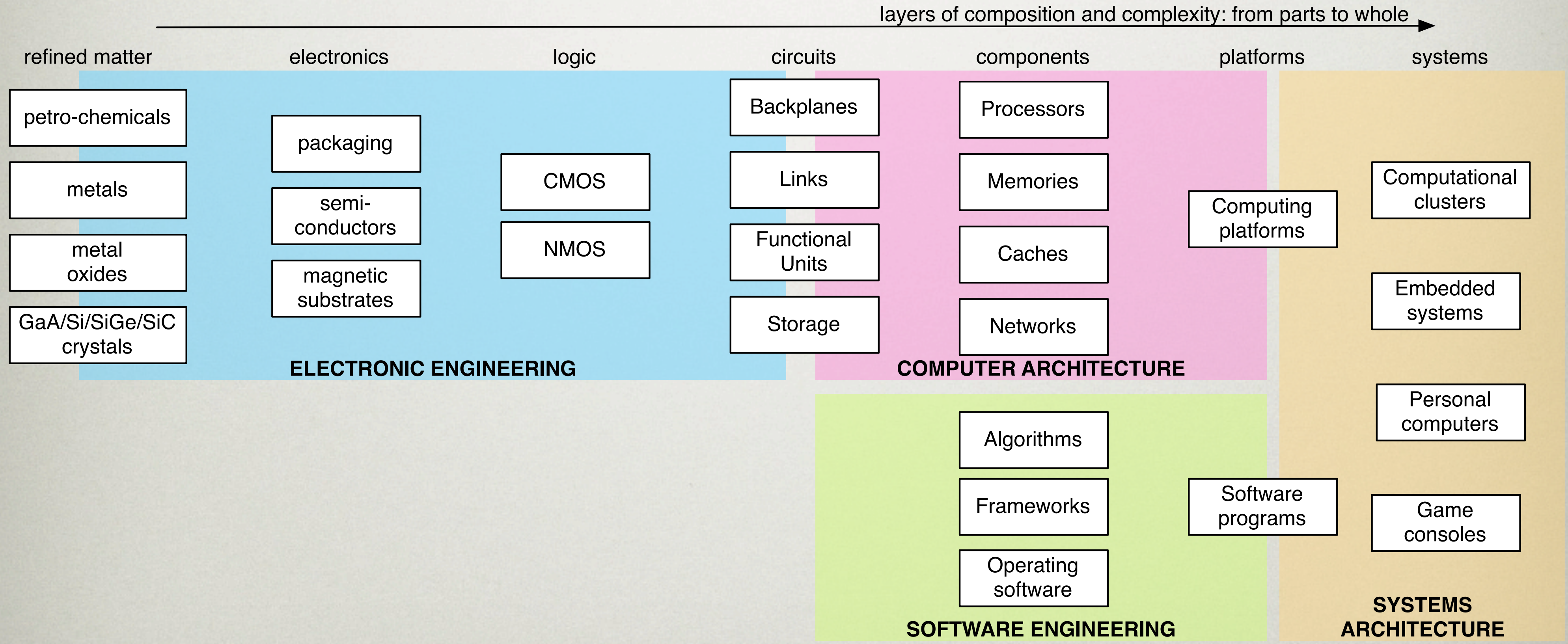
"Ideas vs. realization" - not!

- Common fallacy:
  "coming up with an idea ≠ implementing this idea"
  "Ideas are free, but execution is priceless" – Scott Ginsberg
- In computer architecture:
  - some people *specify* components
    - often using smaller components as sub-parts
  - other people *integrate* designs into systems
  - other people *deploy/validate* systems to applications

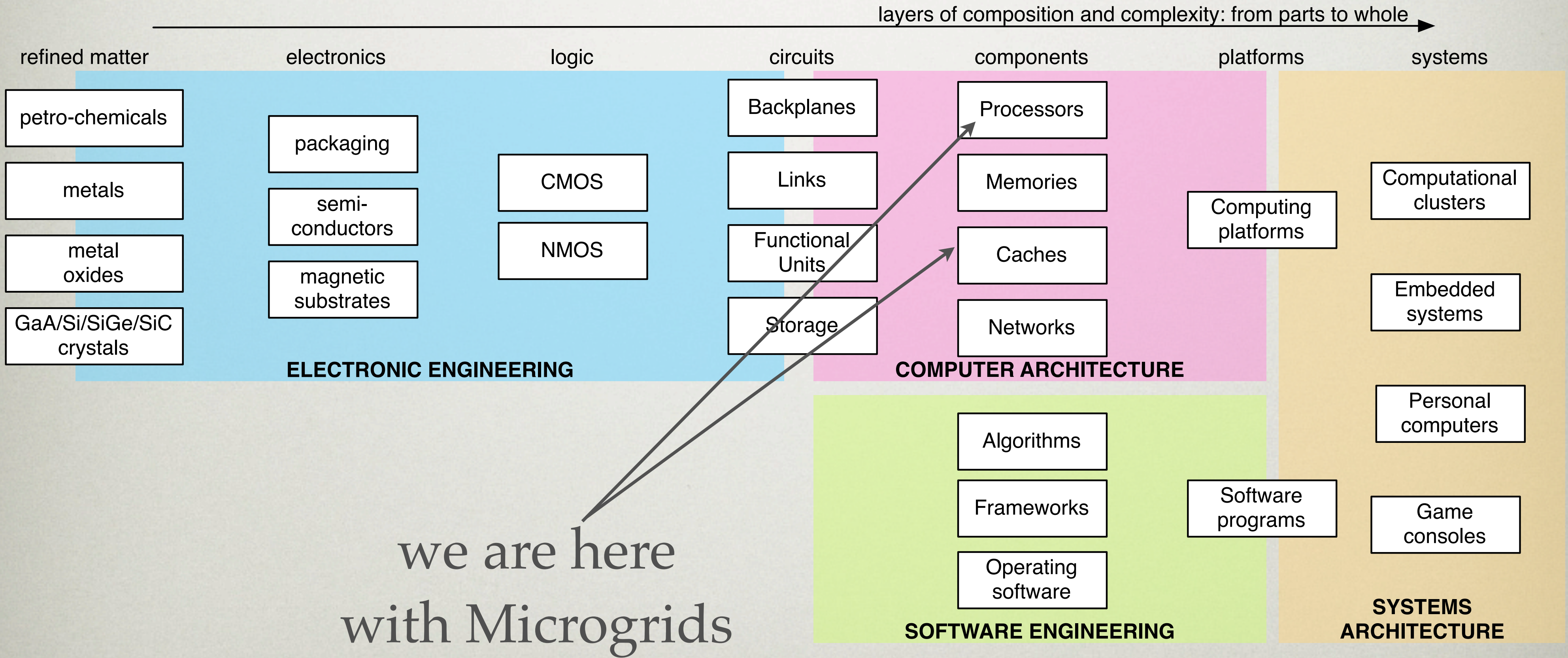
by the way we do that at CSA!

- **Computer architecture is an ecosystem...**
  different people, different responsibilities
- **... in symbiosis with software ecosystems**

COMPUTER ENGINEERING AT A GLANCE



Computer engineering at a glance



Invention vs. application

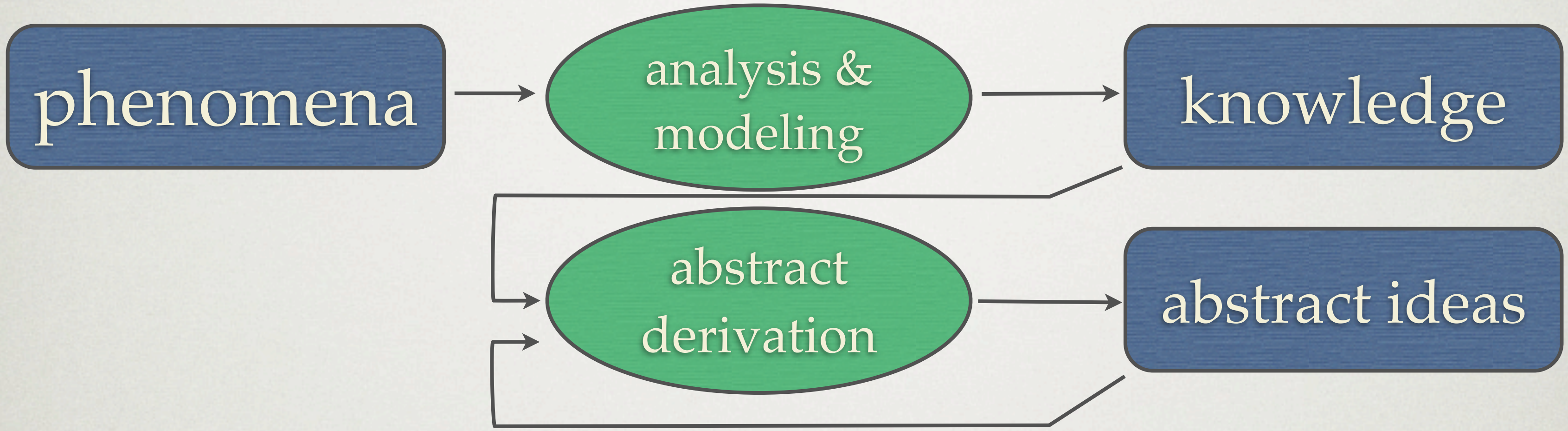
- Two major types of personality profiles
- **"Inventive" types**
  - **creative**, restless
  - *"if I know how it works, it's not interesting any more"*
  - **reward system based on abstraction & variety**
- **"Applicative" types**
  - **dedicated**, focused
  - *"will put hours in it until it works and looks nice"*
  - **reward system based on finished products & fame**

Optimization vs. generation

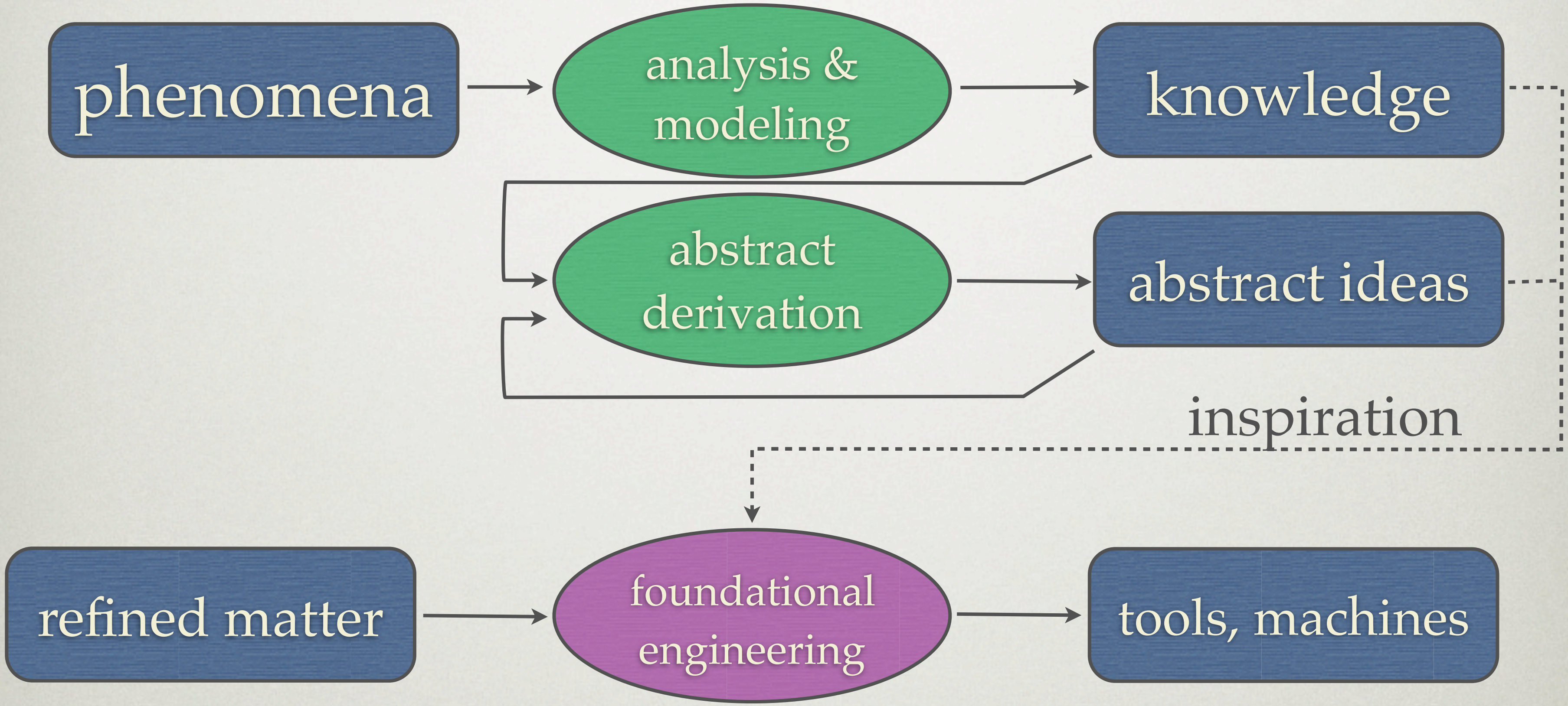
- Two major types of research in **computer architecture**
- **Optimization** – most common
  eg. pipelines, new silicon technology, branch predictors, etc.
  - Mostly uses the **scientific method**
    observe-hypothesise-predict-test-analyze
  - **Incremental**
    new components **comparable** to **previous generations**
- "**Generation**" (for lack of a better word) – less common
  eg. processor registers, RISC, VLIW, hardware multithreading, GPU accelerators
  - Profundly non-scientific: **irrational human creativity**
  - **Disruptive**: not comparable, creates **new areas** for further work
- **Test to distinguish**: can we exploit the outcome **using the same software**?
  - Generation: "no", large software investment necessary to demonstrate

A model for the activities of computer science

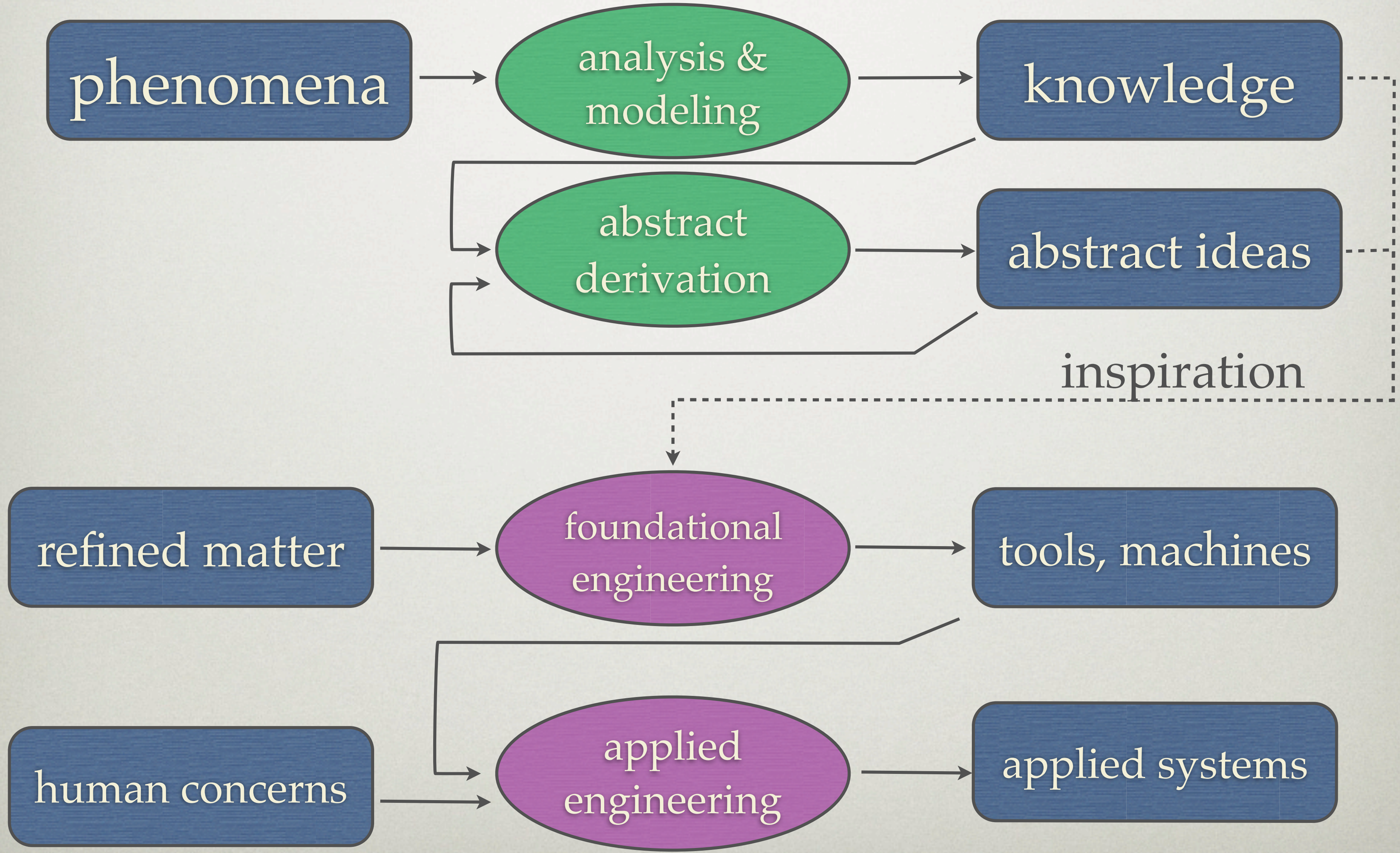
A model for the activities of computer science



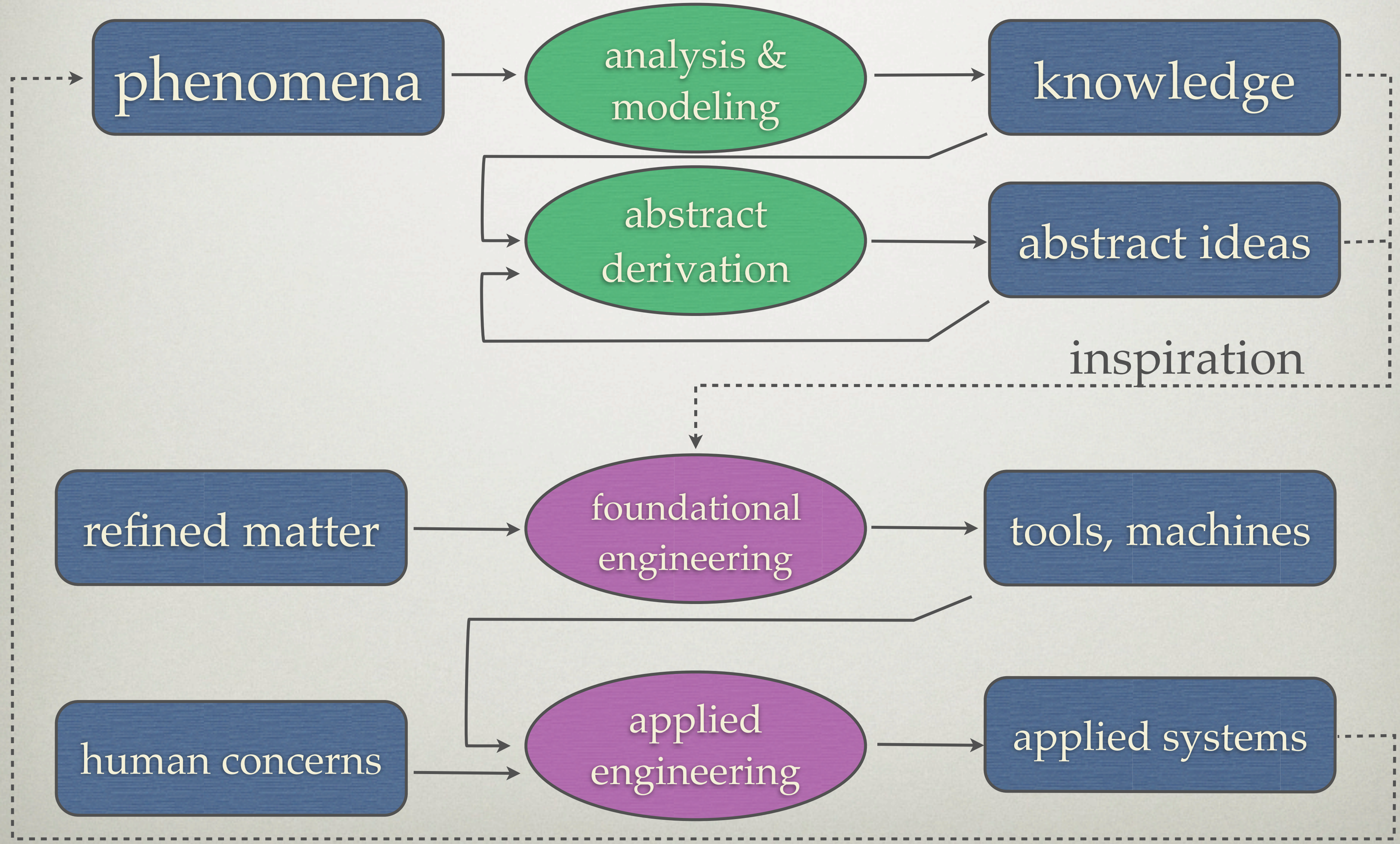
A model for the activities of computer science



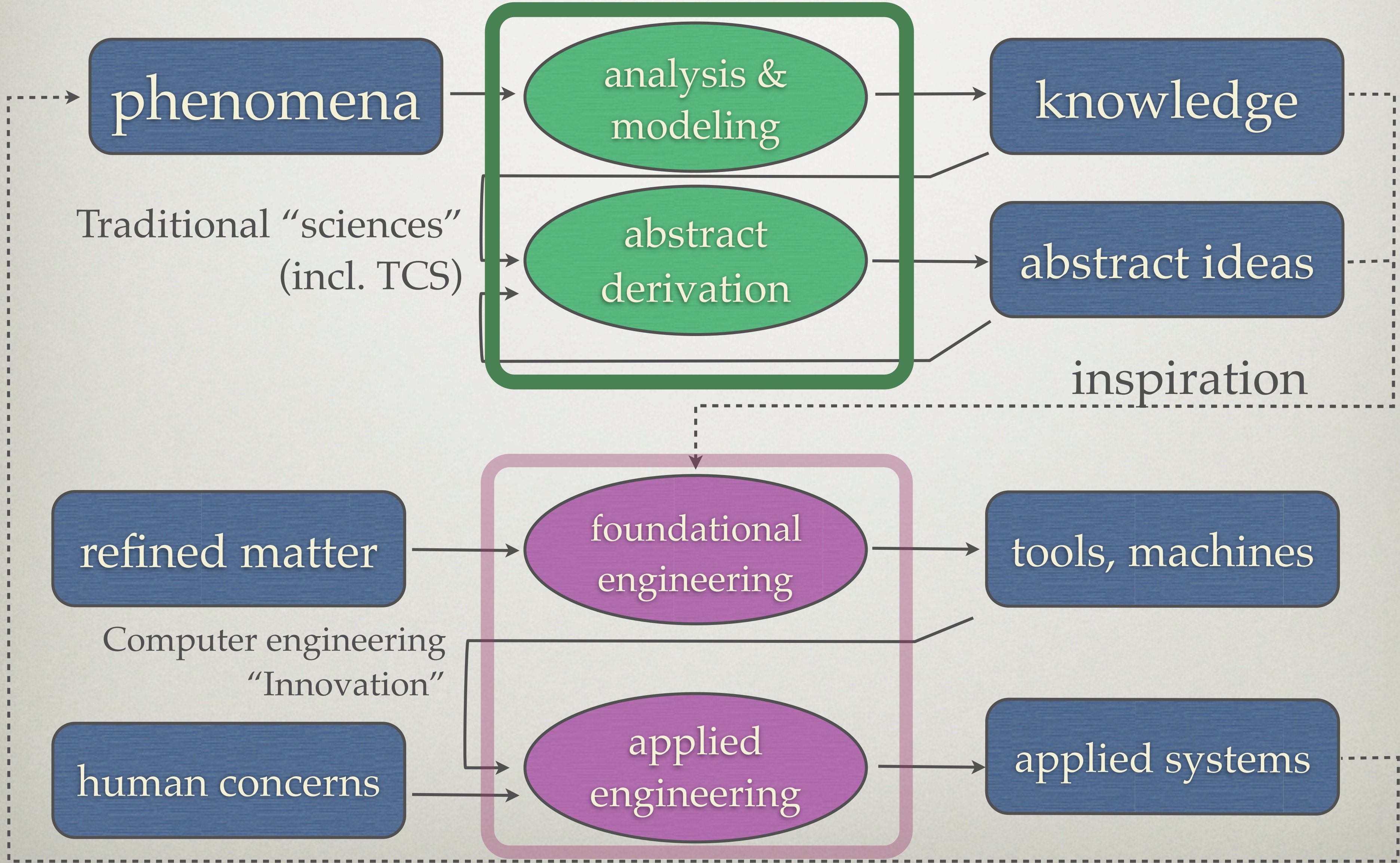
A model for the activities of computer science



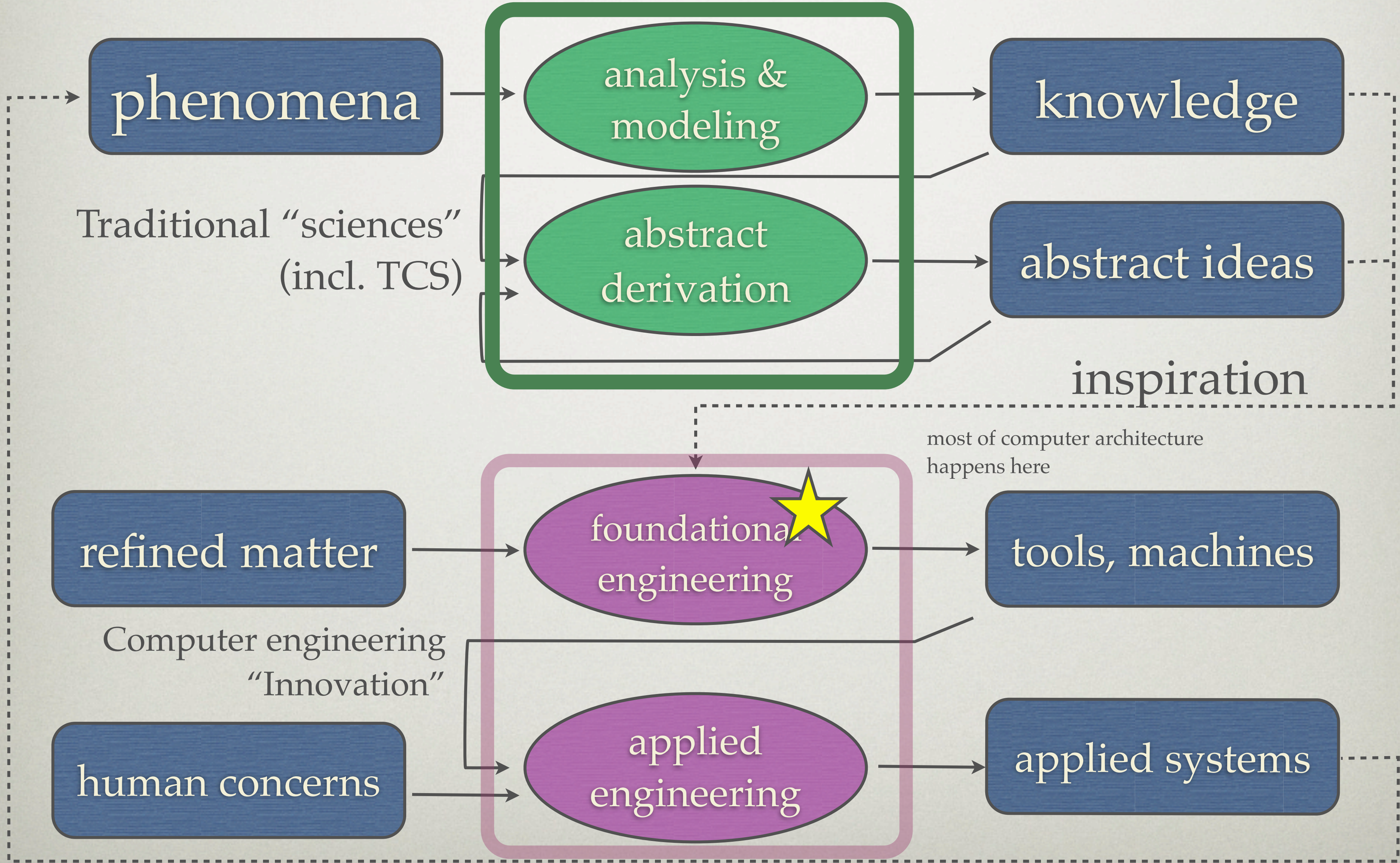
A model for the activities of computer science



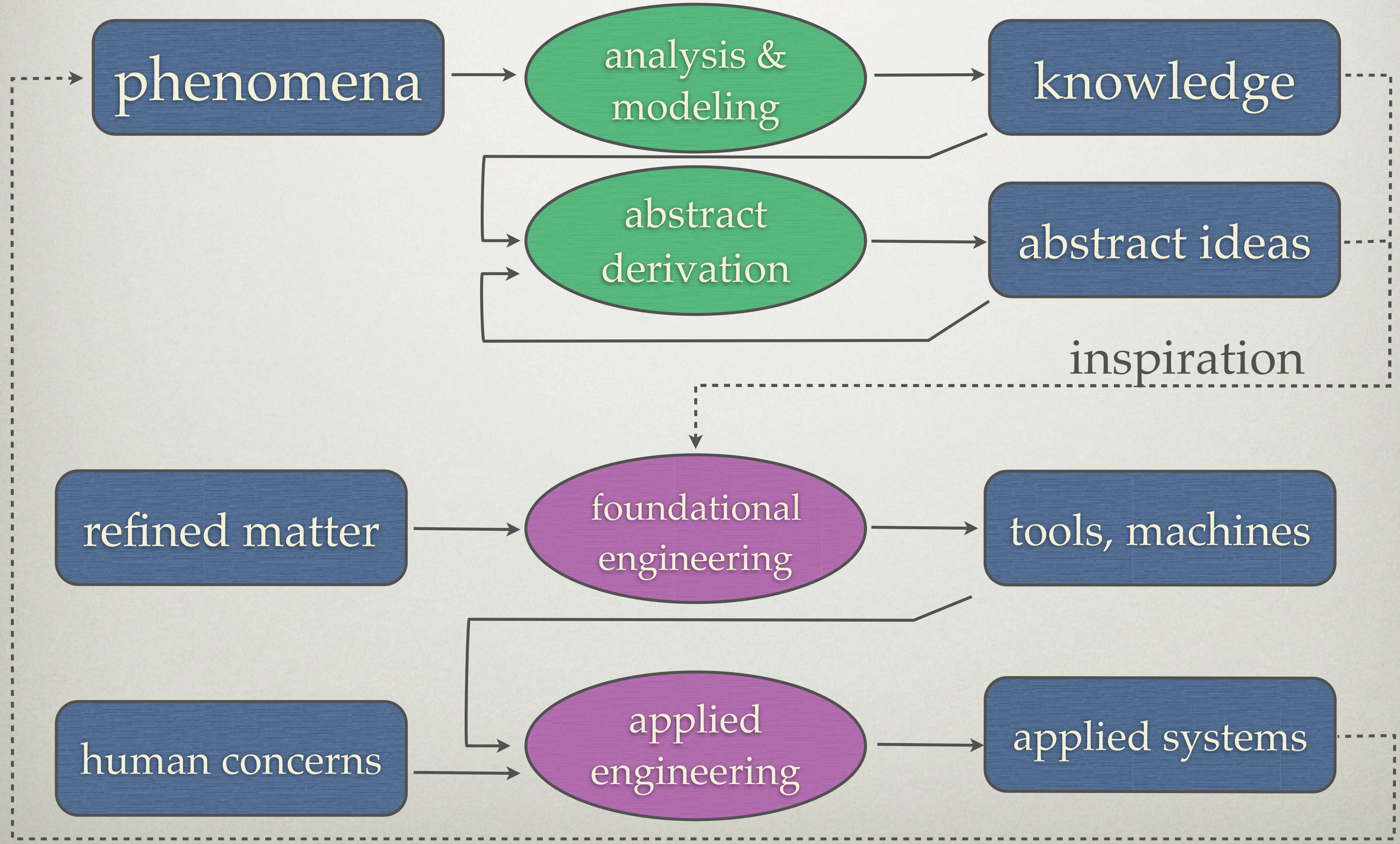
A model for the activities of computer science



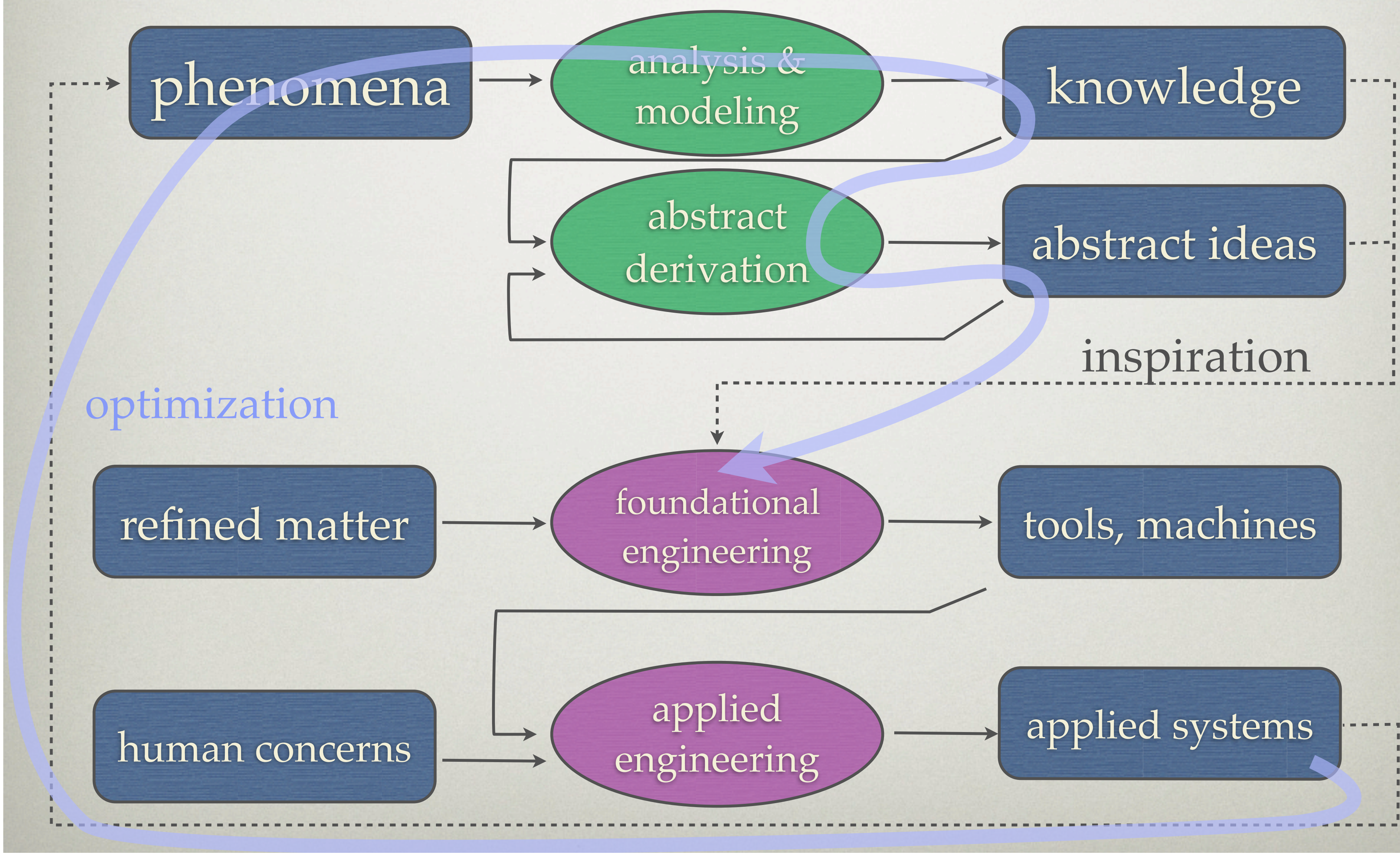
A model for the activities of computer science



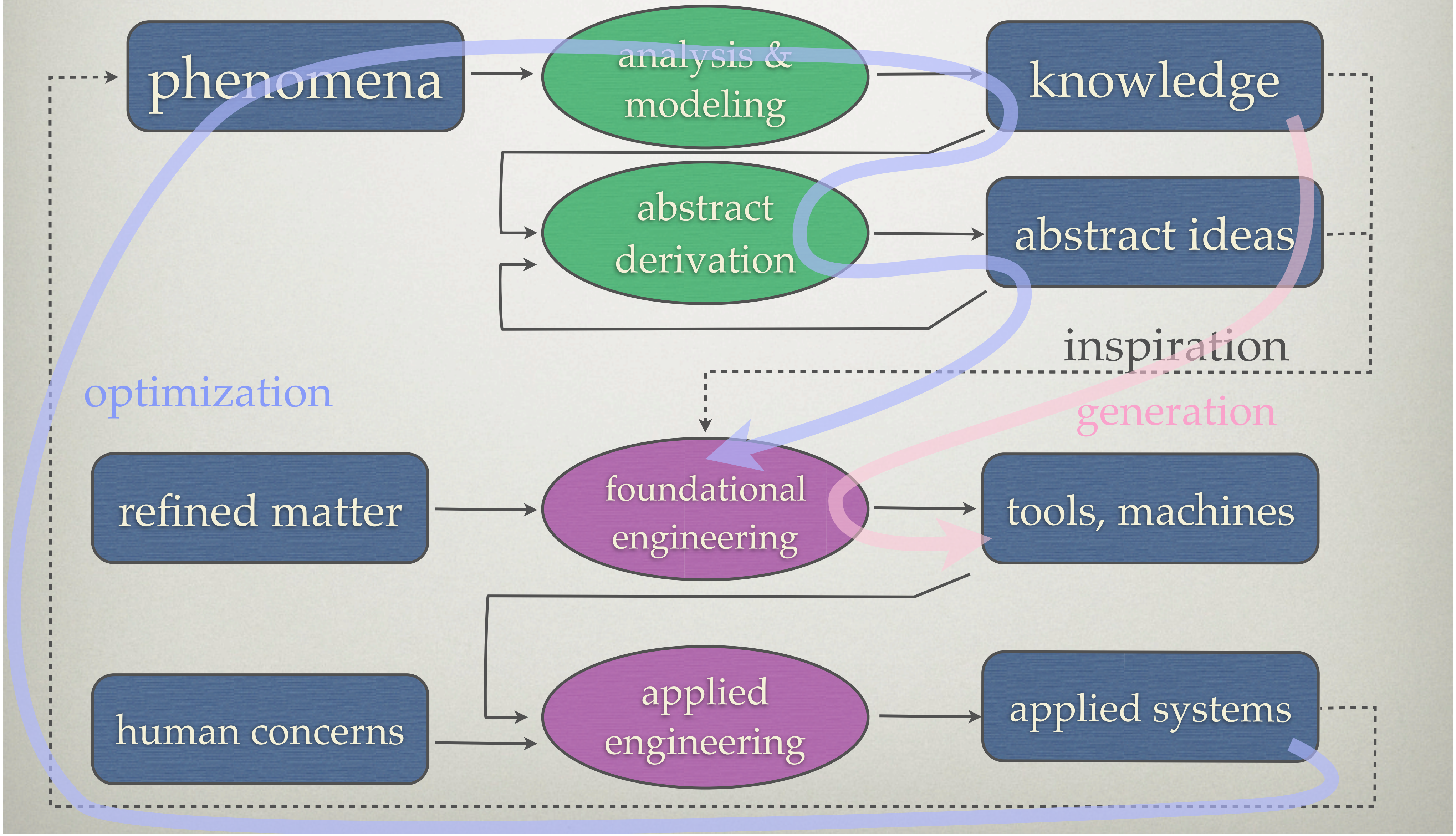
A model for the activities of computer science



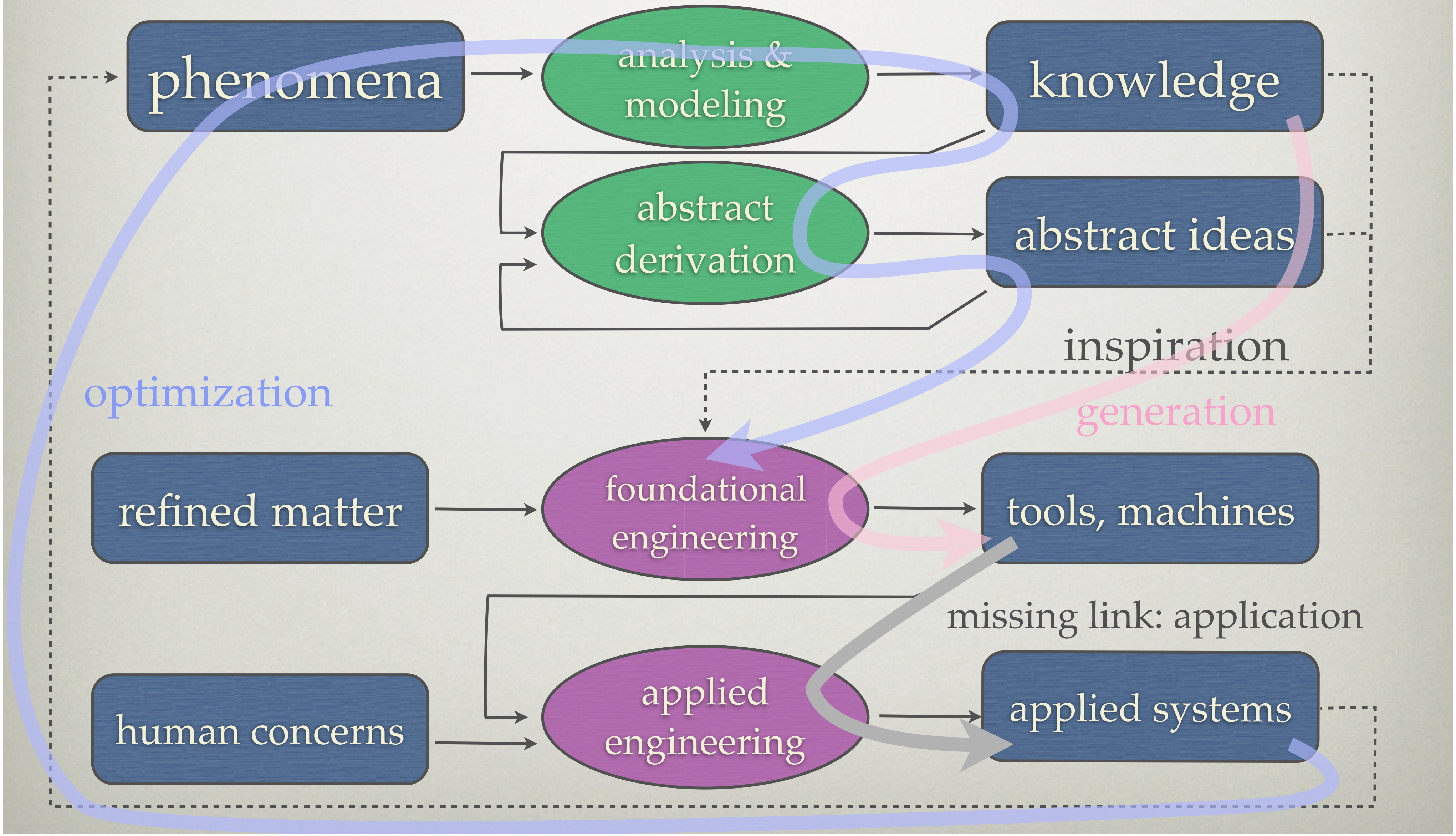
A model for the activities of computer science



A model for the activities of computer science



A model for the activities of computer science



Missing link: from generation to acceptance

- Main issue here becomes **engineering**:
  - **meticulousness** in the realization
    (eg. automated testing, documentation)
  - **scrupulousness** in recognizing and following audiences (customer) expectations
    (eg. check corner cases, provide autonomous demos, provide relevant tutorials)
  - thus **awareness** of how technology fits into the larger picture of a market
- Best done by “applicative” people
- Activity traditionally under responsibility of “**industry**”: private enterprises, high risk but potentially high ROI.

Outcome so far

- "Computer science"
  = computer engineering / "innovation" (**do**)
  inspired/sustained by theoretical computer science (**think**)
- "Innovation"
  = foundational engineering (**invent**)
  + applicative engineering (**make**)
- **Optimization** vs. **generative CS**
  are **different paths** through these activities
- Followed by **different groups of people**
  (different distributions of **personality profiles**)
- In "science organizations" we have an **excess of thinkers and inventors, lack of applicative engineers** – short on "make"

"Science vs. Innovation"
a.k.a.
Politics in computer engineering

Landscape 2005-2015

- **All fields of IT rely on computer engineering**
  - **And so does pretty much everyone's life** at least in the Western world
- **Computers are invented and made by humans**, not nature or other computers
- **There are currently HUGE challenges in computer architecture – likely not solvable with optimization only**
- **Who** will solve these problems?
  How can we **facilitate generative CS?**

Computer architecture enablers

- Things innovators must do to succeed in computer architecture:
  - **a priori analysis & modeling** of system behavior
  - develop **complex & computationally expensive experiments** towards validation of new/optimized computers
  - **specify components**, implement **simulators**
  - for generative CS, implement new **software infrastructure** (libraries, operating systems, compilers)
  - **rewrite and re-run programs** written by other people and see how to make them run "better"
- In other words the people in charge must be both
  - **competent scientists**
  - and **seasoned system and software engineers**

The challenge of foundational CS

- Core issue, not specific to comp. arch.: **creation is non-scientific**
  - **not incremental, not falsifiable, not verifiable**
  - **cannot compare at a small scale** with previous generations (cf earlier Lego brick analogy)
  - **need to build larger systems using the invention** to demonstrate/see "what it's good for"
- **success is measurable only in hindsight** – sometimes years afterwards
- **cannot "measure" progress incrementally** – huge management risk
  - Therefore, **no short-term incentive** to promote and facilitate gen. CS
  - Unclear **how to train and reward** the right personality & skills in people

Issues of morals and politics

- The elephant in the room:
  Why not **delegate innovation entirely** to industry?
  Industry is good at applied CS, why not foundational CS too?
- **Morals**: generative CS is increasingly captured behind corporate closed doors (Samsung, Apple, ARM, Intel, ...)
  - our descendants will ask what did we do to foster **openness, transparency & democracy?**
- **Politics**: what should be the role of research organizations?
  Is it only to produce abstract models models and human-tools for corporations?
  - **There is more money to be gotten for new technology** than for academic papers
    – we may want a slice of that!

Partnerships & education (conclusion)

- The "meat" of our job is computer engineering
  - However **our students currently don't have balanced skills**, and seasoned professionals are expensive to hire locally
  - The working strategy so far has been **publicly-funded partnerships**
  - **However public money is "drying up" too**
- Acknowledge that ⇒ **enhance autonomy of public research groups**
  - Acknowledge that **innovation is carried out by different types of people working together**; "think" vs. "invent" vs. "make" is also a matter of **personality**, not only separate skills
  - **Educate "inventors" and "applicative engineers" separately** + **mix and match personality types in research groups**
  - **But all should know how to write system/infrastructure software** – otherwise, no way to demonstrate inventions in architecture
  - ⇒ **room for improvement in the Dutch higher education programs**