A Note on Computer Systems Research

Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software & Architecture Lab.

Seoul National University

Spring 2026

Computer Systems Research

- What is computer systems research?
  - Dealing with everything between hardware and applications
- The capstone of all the computer science principles:
  - Operating systems
  - Computer architecture
  - Computer networks
  - Compilers
  - Programming languages
  - Databases
  - Security
  - Data structures and algorithms, …

Computer Systems Areas

- Parallel and distributed systems
- Embedded systems
- Database management systems
- Networked systems
- Storage systems
- Cloud computing
- Mobile computing
- Ubiquitous or pervasive computing, IoT
- CPS (Cyber-Physical Systems)
- Blockchain
- Machine learning, Deep learning, LLM
- … and you name it!

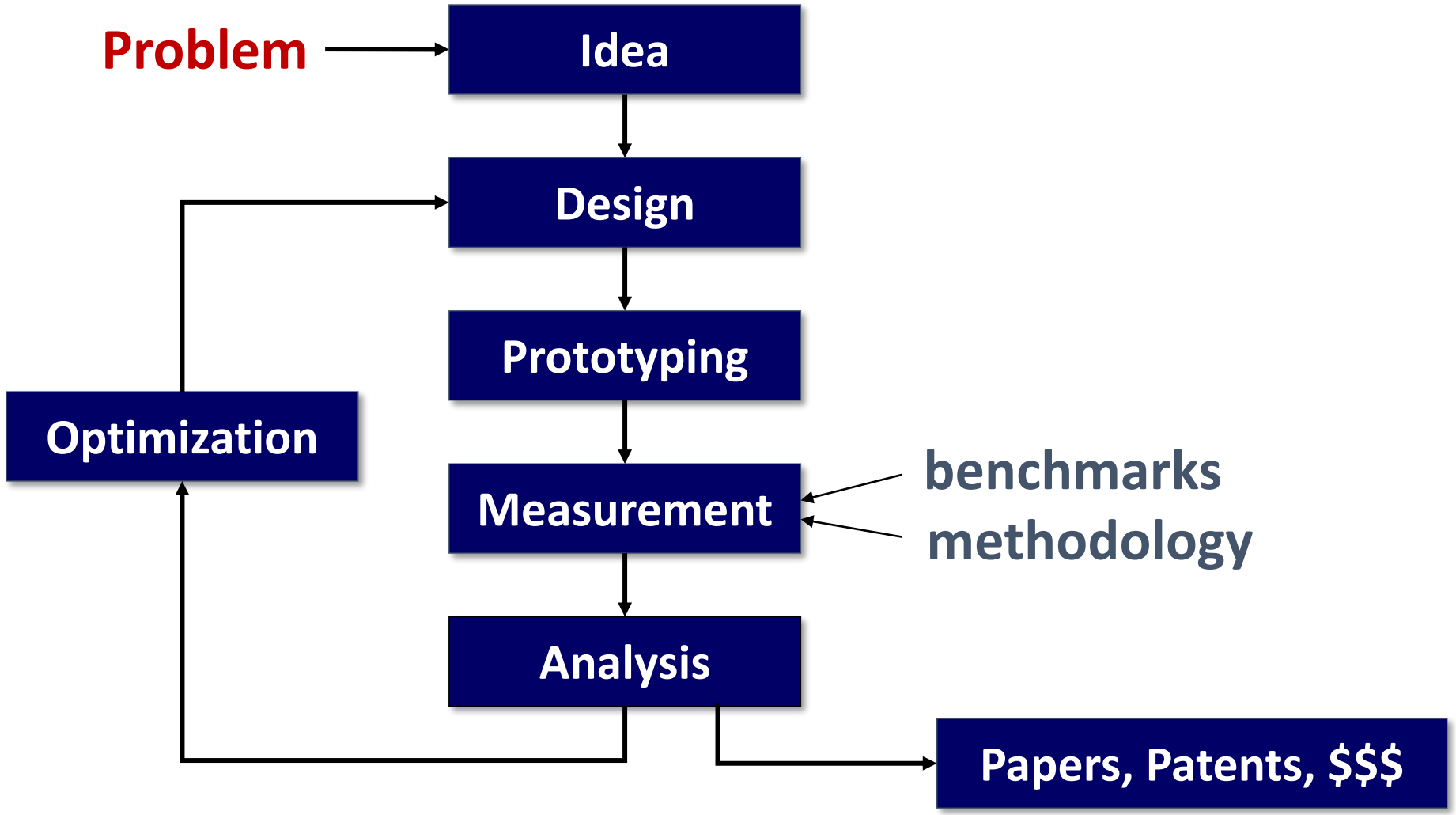
Computer Systems Characteristics

- Designing computer systems is very different from designing an algorithm
- The external requirement is less precisely defined, more complex, and more subject to change
- The system has much more internal structure, hence many internal interfaces
- The measure of success is much less clear

Challenges

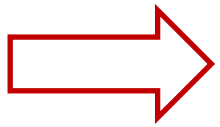
- Large system is difficult to understand or build
- There are not necessarily any right answers
  - There is a sea of possibilities
  - Unclear about how one choice will limit his freedom to make other choices, or affect the size and performance of the entire system
- No one can tell you with certainty that you are right
  - The best way we can do is to avoid choosing a terrible way
  - The art of trade-offs
- You are never done
  - The last 10% to perfection typically consumes 80% of the effort
  - You don't know if the resulting system is perfect or not

Systems Research Cycles



Getting Idea

- Identify a problem/phenomenon in the current systems
  - Physical memories are small
  - CPU is getting faster, but I/O is NOT that much
  - Number of cores is increasing, …
- The world is changing (always):
  - New services: WWW, Web services, P2P, Grid, MMORPG, Cloud, XaaS, YouTube, …
  - New applications: DB, Java, Games, Big Data, Machine learning, Blockchain, …
  - New environments: Wireless, Mobile, Ad-hoc, Battery-powered, VR, AR, 5G, …
  - New architecture/devices: Multi-cores, Flash, SSD, RDMA, GPU, FPGA, NVM, …



Add a functionality
Enhance performance/cost, reliability, or energy-efficiency
Develop a new system

Getting Idea: Some Suggestions

- Have a broad spectrum of knowledge
  - Read widely
  - Participate in seminars (e.g., 즐점세)
  - Don't stick to your *field* (Bad Career Move #1)
- Know the state-of-the-art
  - See what other people are doing
- Stay tuned for emerging technologies and predict their implications
- Be a team player; discuss with others
- There is no free lunch
  - Practice solving problems and building systems whenever you have a chance

Design

- Functionality
  - Does it work?
- Speed
  - Is it fast enough?
- Fault-tolerance
  - Does it keep working?

Source: B. Lampson, "Hints for Computer Systems Design," SOSP, 1983.

Design: Lampson's Hints

Why?	*Functionality* Does it work?	*Speed* Is it fast enough?	*Fault-tolerance* Does it keep working?
Where?			
Completeness	Separate normal and worst case	Shed load End-to-end Safety first	End-to-end
Interface	Do one thing well: Don't generalize Get it right Don't hide power Use procedure arguments Leave it to the client Keep basic interfaces stable Keep a place to stand	Make it fast Split resources Static analysis Dynamic translation	End-to-end Log updates Make actions atomic
Implementation	Plan to throw one away Keep secrets Use a good idea again Divide and conquer	Cache answers Use hints Use brute force Compute in background Batch processing	Make actions atomic Use hints

Source: B. Lampson, "Hints for Computer Systems Design," SOSP, 1983.

Design: Policy vs. Mechanism

- Policy
  - *What* should be done?
  - Policy decisions must be made for all resource allocation and scheduling problems
  - e.g., CPU scheduling: scheduling algorithm, quantum size, priority, etc.

- Mechanism
  - *How* to do something?
  - The tool for implementing a set of policies
  - e.g., CPU scheduling: dispatcher for low-level context switching, priority queues, etc.
  - e.g., X-windows implement "mechanism, not policy"

Design: Separating Policy from Mechanism

- A key principle in operating system design

- Policies are likely to change depending on workloads, and also across places or over time
  - Each change in policy would require a change in the underlying mechanism

- A general mechanism, separated from policy, is more desirable
  - A change in policy would then require redefinition of only certain parameters of the system instead of resulting in a change in the mechanism
  - It is possible to experiment with new policy without breaking mechanisms

Design: Policy / Mechanism Benefits

- Allows to build a more modular system
  - Only need to port mechanisms

- Enables extensible systems
  - Microkernel only implements a basic set of policy free primitives
  - More advanced mechanisms and policies are added via user-created kernel modules or via user programs themselves

- One can design a kernel that allows processes to download their ideal polices directly into the OS

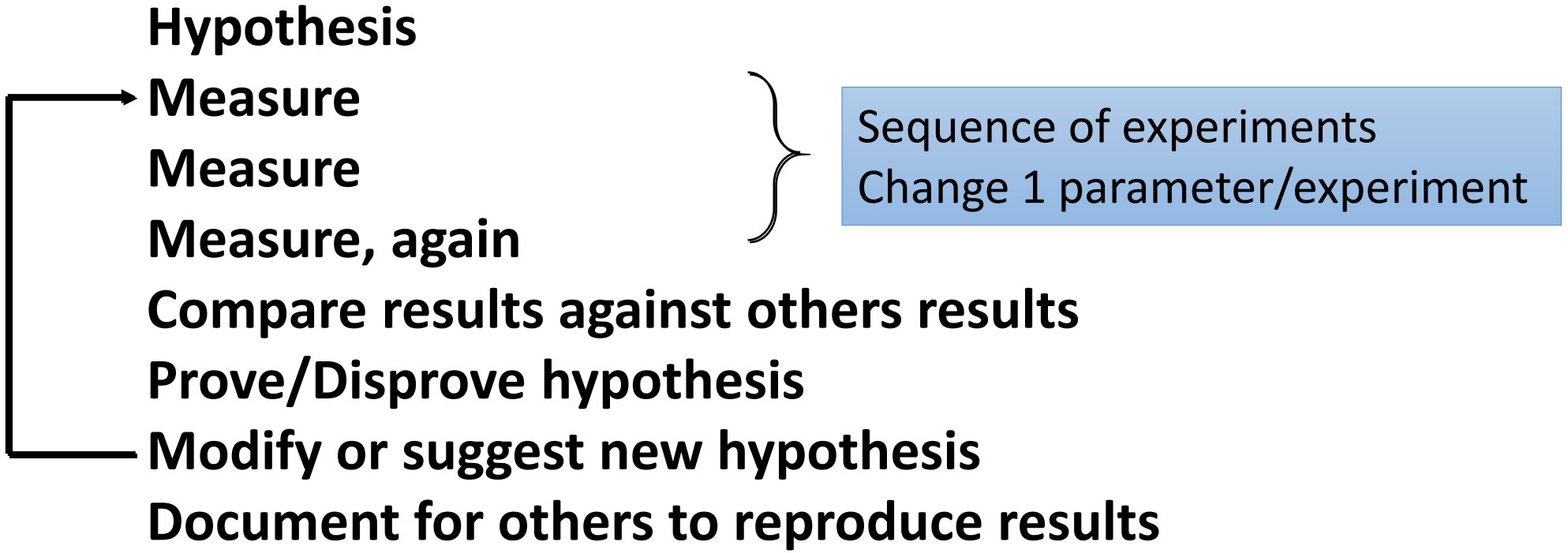
Design: Some Suggestions

- Ask *why* you are doing this
  - Define your problem, goals, and metrics first
  - And then, ask Who? When? What? How?
- List the expected benefits and costs
  - BEFORE any experiments
  - Determine the graphs you would like to see
- Think of any other way you can do better
  - Implementations are expensive; take enough time on design
- Again, be a team player; review your design with others
  - Two heads are better than one

Evaluation: Methodologies

- Use analytical investigations
  - Probably too many assumptions
  - Need to inject real parameters… How to obtain them?
- Use simulations
  - OK if you use a well-known simulator, or open-source your simulator
  - Need to prove your model is correct; run sanity checking experiments to which you know the answer
- Gather traces from real systems
  - Traces are much easier to deal with
- Instrument or modify existing systems: Hard…
- Build prototypes: Harder…

Evaluation: Use Scientific Method



- Use your intuition to *ask* questions, not *answer* them
- What's your metric?

Evaluation: Benchmarking

- Micro-benchmarks? (e.g. fio, dd, …)
  - Useful to test specific features
  - Can evaluate worst-case behavior, isolate features/components
  - But, not representative of "real world" workloads
- Macro-benchmarks? (e.g. filebench, RocksDB/LevelDB, …)
  - Useful to evaluate more realistic workloads
  - But, still synthetically generated
- Trace replay? (e.g. SNIA IOTTA trace repository)
  - Considered most realistic, based on actual system traces
  - But traces can be stale, take long time to replay

Source: E. Zadok, Everything You Always Wanted to Know about Storage Analysis," FAST, 2026.

Writing a Research Paper (1)

- Abstract
  - Introduce area
  - State problem
  - Summarize conclusions (be quantitative)

- Introduction
  - Complete description of problem
  - State more detailed results
  - Short summary of what's have been done
  - Roadmap of the paper

Writing a Research Paper (2)

- Previous work
  - Do a thorough literature search (from 60's)
  - Neither more nor less list of references
  - Relate your work to existing work
  - Demonstrate how your work fits in to the grand scheme of things

- Design & Implementation
  - Do describe your research
  - Be thorough, but concise
  - Clarify why you made such design decisions

Writing a Research Paper (3)

- Evaluation
  - Describe experimental setup (both hardware and software)
  - Explain your methodologies
  - Explain expected results, reasons for such expectation
  - Explain surprising difference
  - Visual presentation of data

- Conclusions
  - State results again
  - State significance of results
  - Tell people what they should have learned

Writing a Research Paper (4)

- Future work
  - What questions still remain
  - What new questions have arisen
  - How your work can be extended
- General tips
  - Spell check: very important
  - Grammar check: very difficult for us, but try
  - Style check: passive voice, plural, he/she, …
  - Consistency in figures, graphs, references, etc.
  - As research is going on, start writing and write often: very important
  - Let the paper sit for a few days before proofreading
  - Read many papers!

Evaluation Criteria for Systems Research

- Importance of a problem
- Soundness of assumptions
- Originality and novelty of ideas
- Availability of real implementations
- Exploration of alternative design choices
- Value of lessons learned

Systems Research is Multifaceted

- Science
  - Discover truth by scientific methods
  - Identify the effects and limitations of computer artifacts on the physical world
- Engineering
  - Achieve a solution that works for a particular problem
  - Develop "good enough" heuristics
- Art
  - Elegance, beauty, and simplicity
  - Make complex ideas more palatable or more comprehensible

Source: A. Brown et al., "The Many Faces of Systems Research – And How to Evaluate Them," HotOS, 2005.

References

- B. W. Lampson, "Hints for Computer Systems Design," SOSP, 1983.
- J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end Arguments in Systems Design", ICDCS, 1981.
- R. Levin and D. D. Redell, "An Evaluation of the Ninth SOSP Submissions or How (and How Not) to Write a Good Systems Paper," ACM Operating Systems Review, 1983.
- D. A. Patterson, "How to Have a Bad Career in Research/Academia," CRA Academic Careers Workshop, 2002. (https://www.youtube.com/watch?v=Rn1w4MRHIhc)
- A. B. Brown, A. Chanda, R. Farrow, A. Fedorova, P. Maniatis, and M. L. Scott, "The Many Faces of Systems Research – And How to Evaluate Them," HotOS, 2005.
- E. Zadok, "Everything You Always Wanted to Know about Storage Analysis (But Were Afraid to Ask ;)", FAST, 2026.