S

S is the field with the mundane name Systems
—with op'rating, distributed, fault-tolerant items.
It's expanded and has lots of networking problems.
CS at Cornell is one place that does solve 'ems.

One of our enduring strengths is the ability to marshal a broad, sustained response over decades, harnessing skills in both theory and practice. Many building blocks used in distributed systems trace back to our research, like the fail-stop processor abstraction, fault-tolerant broadcast, state machine replication, virtual synchrony, and failure detectors. Our products may not be huge, but the ideas used in them are. So, if you are considering a PhD in systems, think of working with:

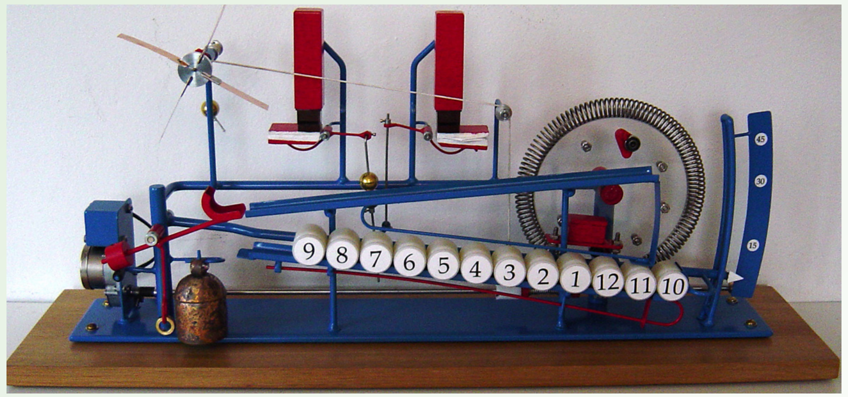
Ken Birman, whose ISIS Toolkit runs the NY and the Swiss stock exchanges and whose ISIS-style process-group replication influenced the CORBA fault-tolerance standard.

Fred Schneider, who, with his grad student, defined safety and proved that any program property could be decomposed into a safety and a liveness property and who has been influential in fault-tolerant distributed systems including fail-stop processors and hypervisor-based fault tolerance.

Van Renesse and Birman, whose information management system Astrolabe is being used by Amazon and whose systems Horus and Ensemble influenced IBM and Microsoft products.

Gün Sirer, whose CoDoNS system has been deployed to serve the Internet domain name space for all of China.

Paul Francis, who developed Network Address Translation (NAT), which staved off the IP shortage crisis and is used in the router that hooks your home computers together, and who influenced the new Internet standard routing protocol IPv6.


This cuckinetic-clock system was designed by George Rhoads, a painter-sculptor, who also designs rolling ball machines and wind sculptures. George lives in Ithaca. See www.georgerhoads.com.

T

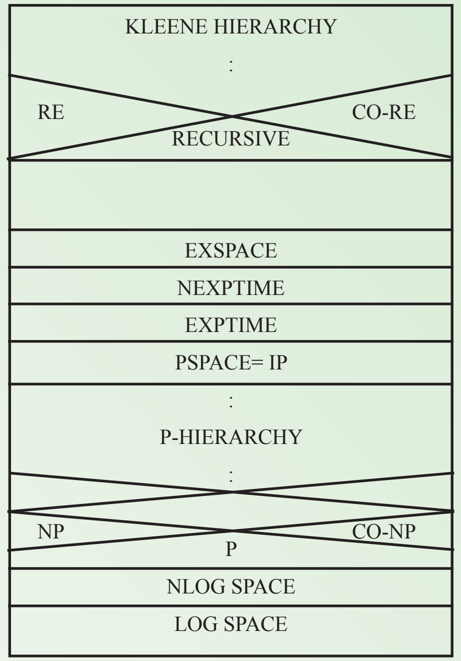
T's for Computing —the Theory thereof.
Our first chair Hartmanis started it off.
Profs at Cornell helped push it aloft,
Like Constable, Kozen, and John E. Hopcroft.
But Theory is not simply what those profs do,
It's a tool, a way of thinking for me and you.
When understanding and advance would appear very tough
Theory provides the path through the rough.

The year Juris Hartmanis became Chair of our new CS department, 1965, he published a paper with Richard Stearns that introduced a new field and gave it its name: Computational Complexity. That, together with John Hopcroft's (joined in 1967) work in automata, formal languages, and algorithms and Bob Constable's (1968) work in subrecursive function theory and logic and mathematical reasoning established Cornell as one of the best places for theoretical work in computing.

In the early days, most faculty in CS believed in the use of theory to make advances, when it made sense to do so —be it programming language design, semantics, operating systems, information retrieval, or numerical analysis— and this contributed to our long-lasting cohesiveness.

This understanding of the use of theory continues today. Look, for example, at Bart Selman, an AI guy, who worked with physicists to unearth phase transitions, like water freezing, in certain instances of the Satisfiability problem in computational complexity. In data mining, the science of networks, language-based security, mission critical systems —in these fields and more, our strengths can be traced to theoretical and practical people working side by side, or to people who refuse to be called "theoretical" or "practical" because they are both.

Our undergrad majors get a heavy dose of theory of computing, and it pays off. We continue to hear from those that go on to grad school in CS that others struggle with theory while they have no problem.


KLEENE HIERARCHY
:
RE
CO-RE
RECURSIVE
EXSPACE
NEXPTIME
EXPTIME
PSPACE= IP
:
P-HIERARCHY
:
NP
CO-NP
P
NLOG SPACE
LOG SPACE