Rattle
Simpler builds for smaller use cases

Neil Mitchell
https://ndmitchell.com/
Build two C files and link

```bash
$ cat make.sh
gcc -c main.c
gcc -c util.c
gcc -o main.exe main.o util.o
```
Shell script

- Simple to write
- Full control over commands

Build system

- More complex
- Must specify dependencies
  - E.g. header files, toolchain

But you gain:

- Parallelism
- Incrementality
Introducing Rattle

$rattle$ make.sh

Gives you parallelism, incrementality, cloud builds.

https://github.com/ndmitchell/rattle

Build Systems with Perfect Dependencies,
Sarah Spall, Neil Mitchell and Sam Tobin-Hochstadt
OOPSLA 2020
How to get incrementality?

- The script runs a series of commands
  - The future commands can depend on the result of previous commands (dynamic dependencies)
- For each command, Rattle records the inputs/outputs using fsatrace
  - Syscall hooking, LD_LIBRARY_PRELOAD, Windows hooks
- Next time it encounters that command, if no inputs have changed, the outputs are reused
  - Assumes commands are deterministic

Fabricate was one of the first build systems to do this trick.
How to get cloud builds?

- Whenever we run a command, we store the inputs/outputs in a cloud cache
- Before running a command, if any command matches, download the outputs

Not quite as simple as it seems… Some inputs (e.g. C files) may change which other inputs are required (e.g. header files). But (at worst) just scan for a match.
How to get parallelism?

The tricky one!

- Guess what commands will come next. Run them. See if you were right.
- Speculation - think of the CPU speculating on instructions
  - And remember how that has turned out - lots of tricky details
- For speculation to be valid, we need to know certain properties about commands
  - E.g. doesn’t read a file that hasn’t yet been written
  - The paper introduced “hazards” and proves the necessary properties, Rattle checks them
  - If hazards trip you up, rerun (speed hit)
Does it work? FSATrace

Same time as Make, despite not having the commit info.
Does it work? Node.js

Faster than make, because dependencies are precise
Why “small” use cases?

- Immature technology (technology preview really)
- Must give a single linearisable trace
  - Doing that *compositionally* at scale often requires dependencies

Rattle makes it easy to do a simple build system.

Sweet spot might be small open-source multi-language projects?