accu conference 2024

Magic Powers at Your Fingertips

Boosting your Developer's Mojo with GDB

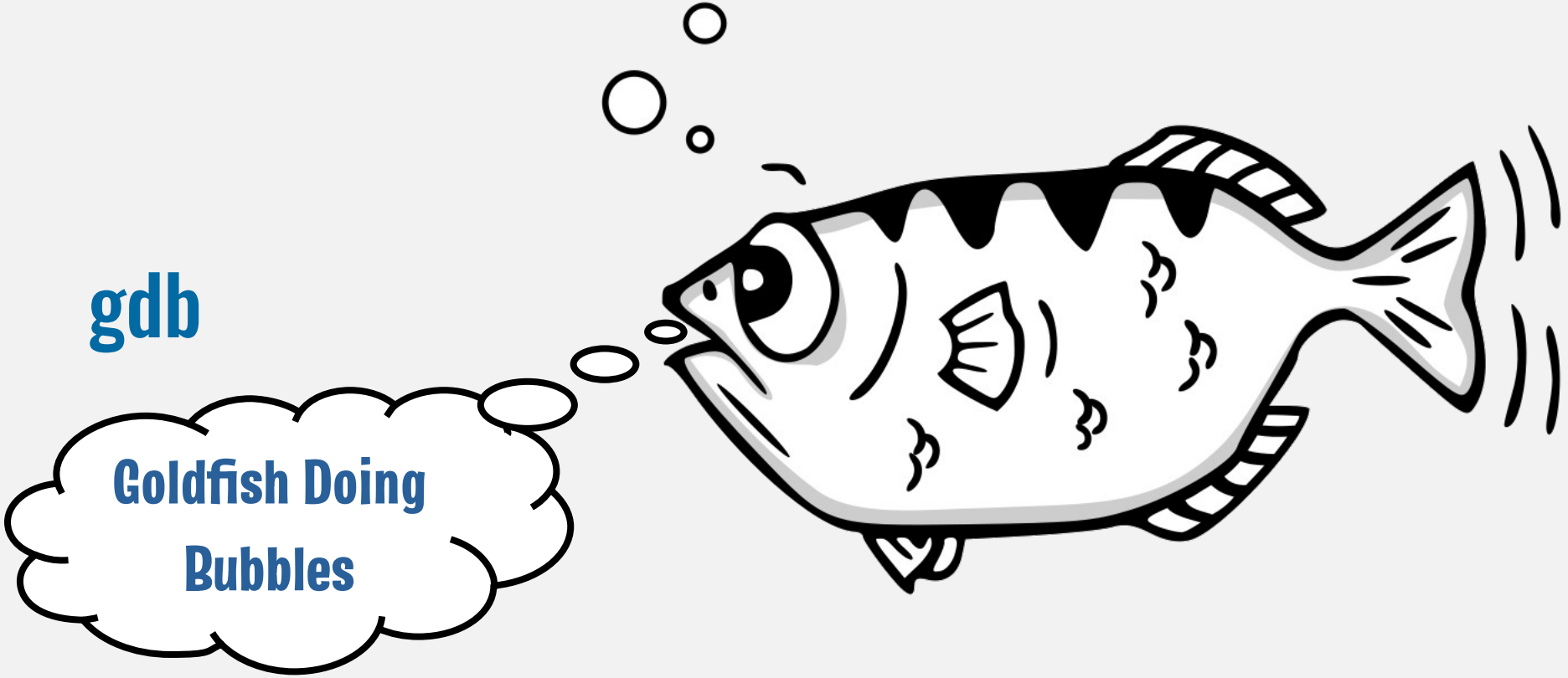
Dawid Zalewski

Magic powers at your fingertips

boosting your developer's mojo with gdb

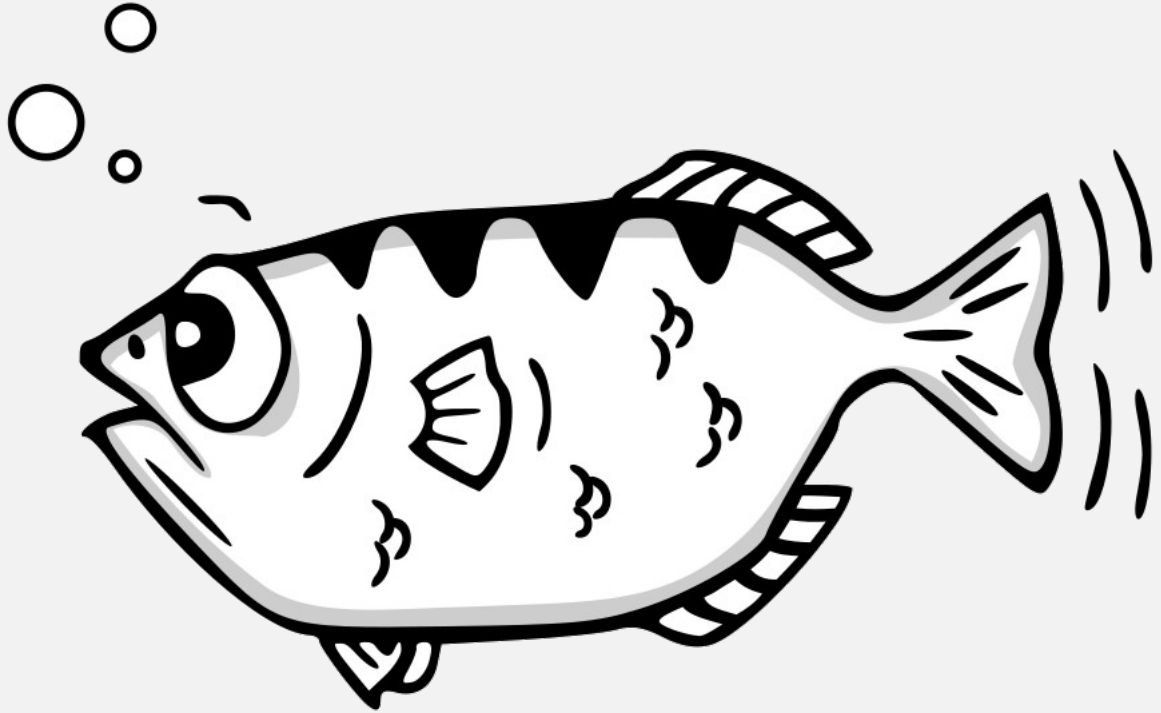
Dawid Zalewski

gdb



gdb logo ©Jamie Guinan & Andreas Arnez CC BY-SA 3.0 US

gdb
(the GNU debugger)



gdb logo ©Jamie Guinan & Andreas Arnez CC BY-SA 3.0 US

debugging (noun)

the process of identifying and removing errors from computer hardware or software

/*Oxford Dictionary*/

debugging (noun)

an emergent activity that arises when the mental model of a program or its data differs from its observed behavior

/* me */

bash

```
int main() {
  std::array numbers{ 0017,
                      0025,
                      1764 };



  return numbers[2] / (numbers[0] + numbers[1]);
}

$ g++ -Wall -Wextra -pedantic -std=c++23 main.cpp -o main.a
$ ./main.a ; echo $?
49
```

bash

```
int main() {
  std::array numbers{ 0017,
                      0025,
                      1764 };

  std::print("Result: {}", numbers[2] / (numbers[0] + numbers[1]));

  return numbers[2] / (numbers[0] + numbers[1]);
}

$ g++ -Wall -Wextra -pedantic -std=c++23 main.cpp -o main.a
$ ./main.a ; echo $?
Result: 49
49
```

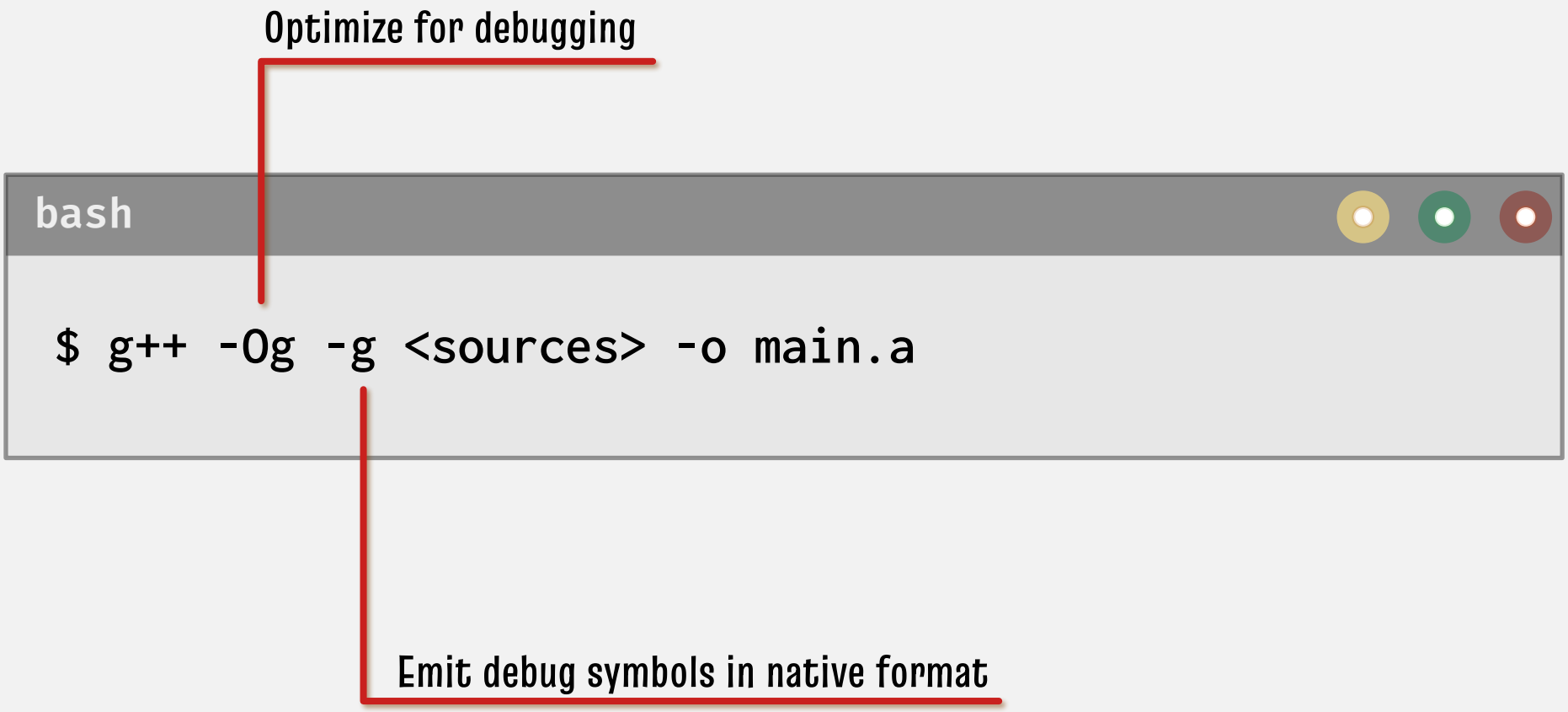
debugging (noun)

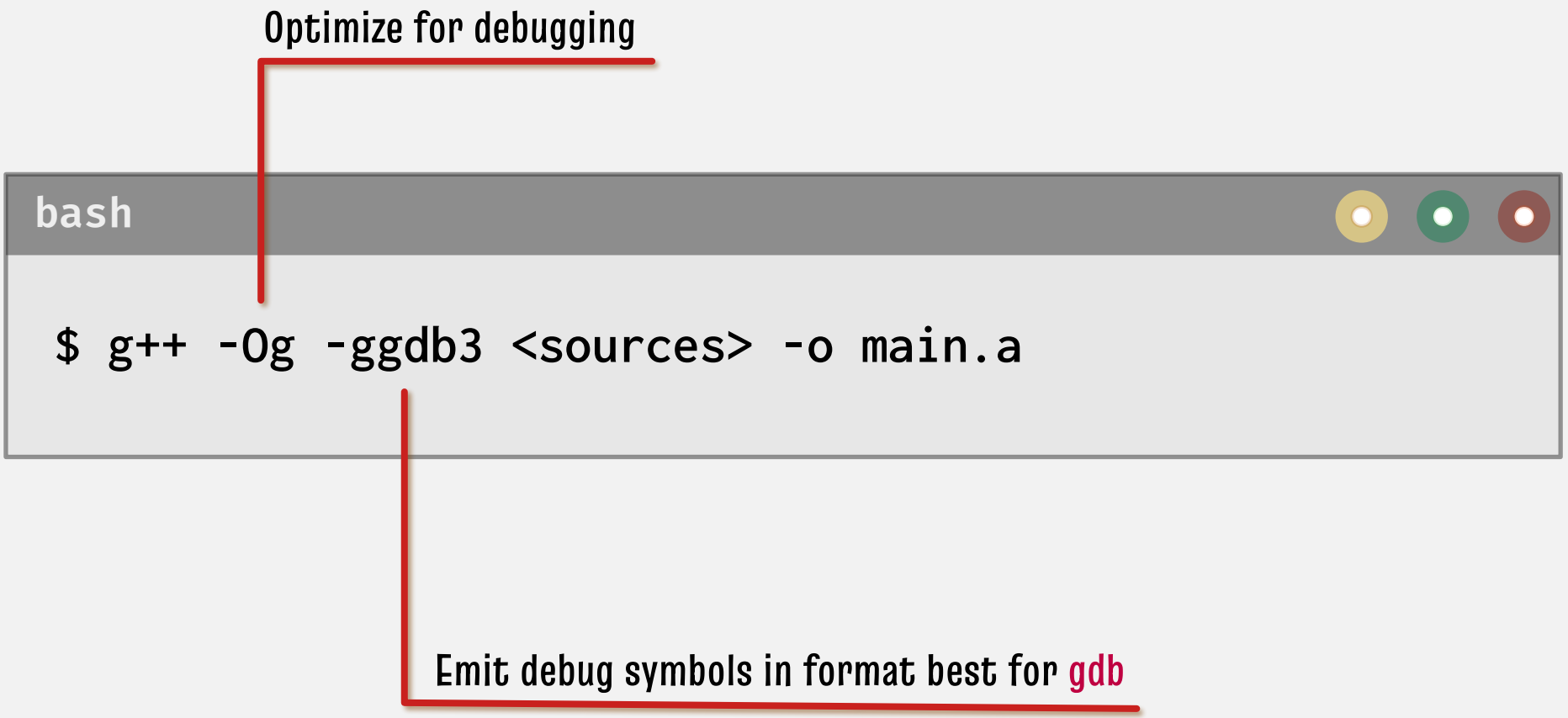
an emergent activity that arises when the mental model of a program or its data differs from its observed runtime behavior, and the underlying reason cannot be easily found

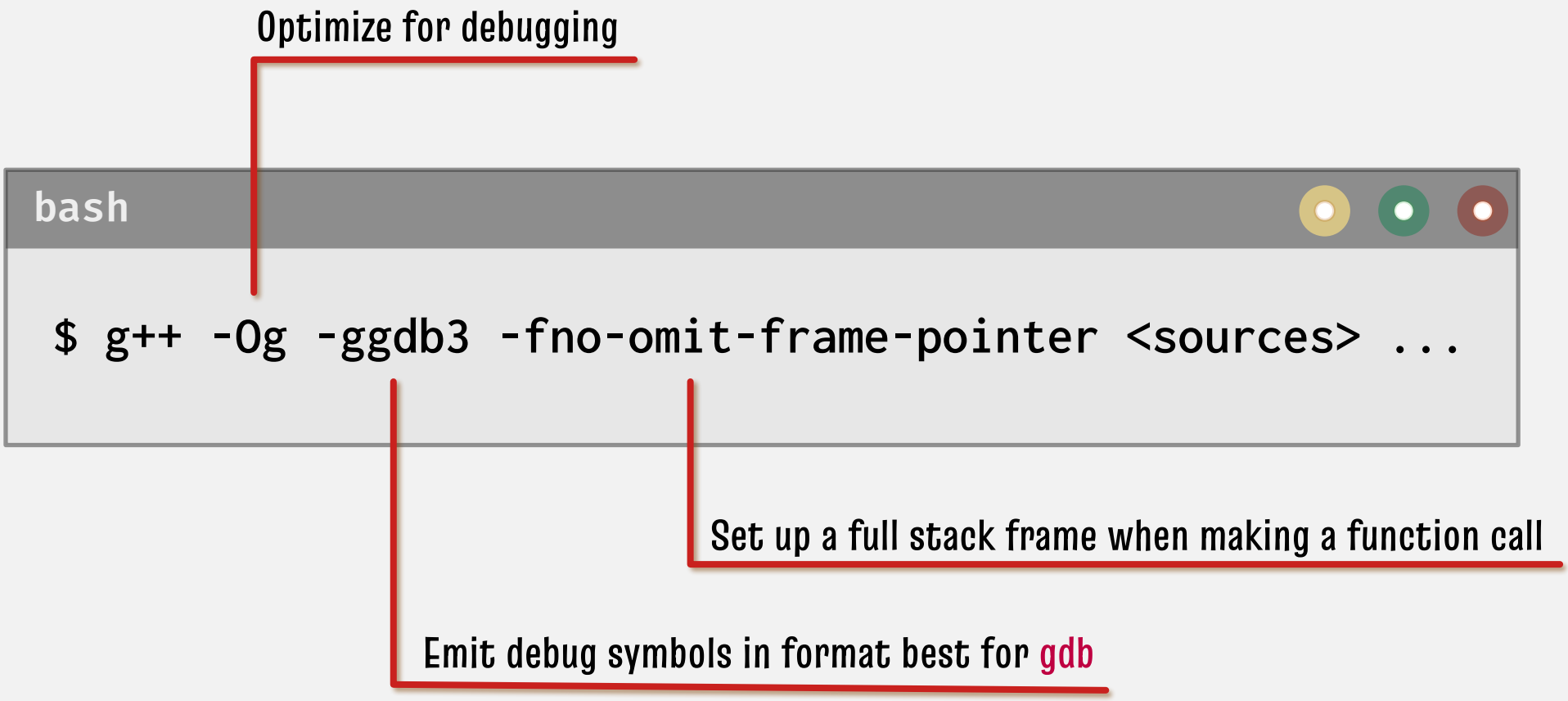
/* me */

bash | gdb

```
$ g++ -Wall -Wextra -pedantic -std=c++23 main.cpp -o main.a
$ gdb -q main.a
Reading symbols from main.a...
(No debugging symbols found in main.a)
(gdb)
```

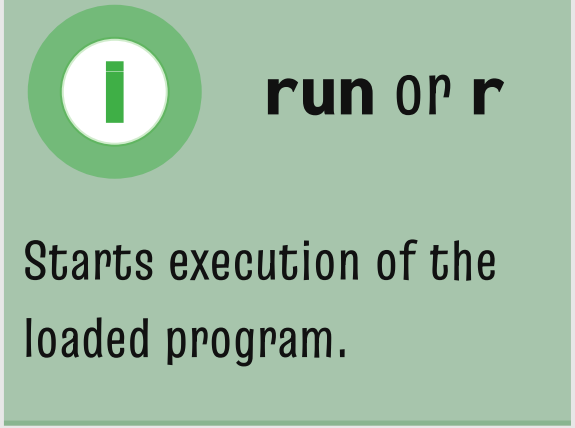

Optimize for debugging
bash
$ g++ -Og -g <sources> -o main.a
Emit debug symbols in native format


Optimize for debugging
bash
$ g++ -Og -ggdb3 <sources> -o main.a
Emit debug symbols in format best for gdb


Optimize for debugging
bash
$ g++ -Og -ggdb3 -fno-omit-frame-pointer <sources> ...
Set up a full stack frame when making a function call
Emit debug symbols in format best for gdb

bash | gdb

```
$ g++ -Wall -Wextra -pedantic -std=c++23 -ggdb3 -Og main.cpp -o main.a
$ gdb -q main.a
(gdb) run
Starting program: ./main.a
[Thread debugging using libthread_db enabled]
Using host libthread_db library
"/lib/x86_64-linux-gnu/libthread_db.so.1".
[Inferior 1 (process 6485) exited with code 061]
(gdb)
```



bash | gdb

```
(gdb) start
Temporary breakpoint 1 at 0x11bf: file ./main.cpp, line 4.
Starting program: ./main.a
[Thread debugging using libthread_db enabled]
Using host libthread_db library
"/lib/x86_64-linux-gnu/libthread_db.so.1".

Temporary breakpoint 1, main () at ./main.cpp:4
4       int main() {
(gdb)
```

start

Starts execution of the program, breaking at the entry point.

bash | gdb

```
(gdb) start
Temporary breakpoint 1 at 0x11bf: file ./main.cpp, line 4.
Starting program: ./main.a
[Thread debugging using libthread_db enabled]
Using host libthread_db library
"/lib/x86_64-linux-gnu/libthread_db.so.1".

Temporary breakpoint 1, main () at ./main.cpp:4
4       int main() {
(gdb) next
5           std::array numbers{
(gdb)
13          return numbers[2] / (numbers[0] + numbers[1]);
(gdb)
```

next or **n**

Continues execution

to the next line of code.

bash | gdb

```
(gdb) start
4       int main() {
(gdb) next
5           std::array numbers{
(gdb)
13          return numbers[2] / (numbers[0] + numbers[1]);
(gdb) print numbers._M_elems
$1 = {13, 23, 1764}
(gdb) p $1[0] + $1[1]
$2 = 36
(gdb) p $1[2] / 42
$3 = 42
(gdb) p $1[2] / $2
$4 = 49
```

print or **p**

Evaluates an expression and prints the result.

bash | gdb

```
(gdb) start
4       int main() {
(gdb) next
5               std::array numbers{
(gdb)
13              return numbers[2] / (numbers[0] + numbers[1]);
(gdb) print numbers._M_elems
$1 = {13, 23, 1764}
(gdb) p $1[0] + $1[1]
$2 = 36
(gdb) p $1[2] / 42
$3 = 42
(gdb) p $1[2] / $2
$4 = 49
(gdb) list main
```

list or **l**

List the source code around <location>

text | basic commands

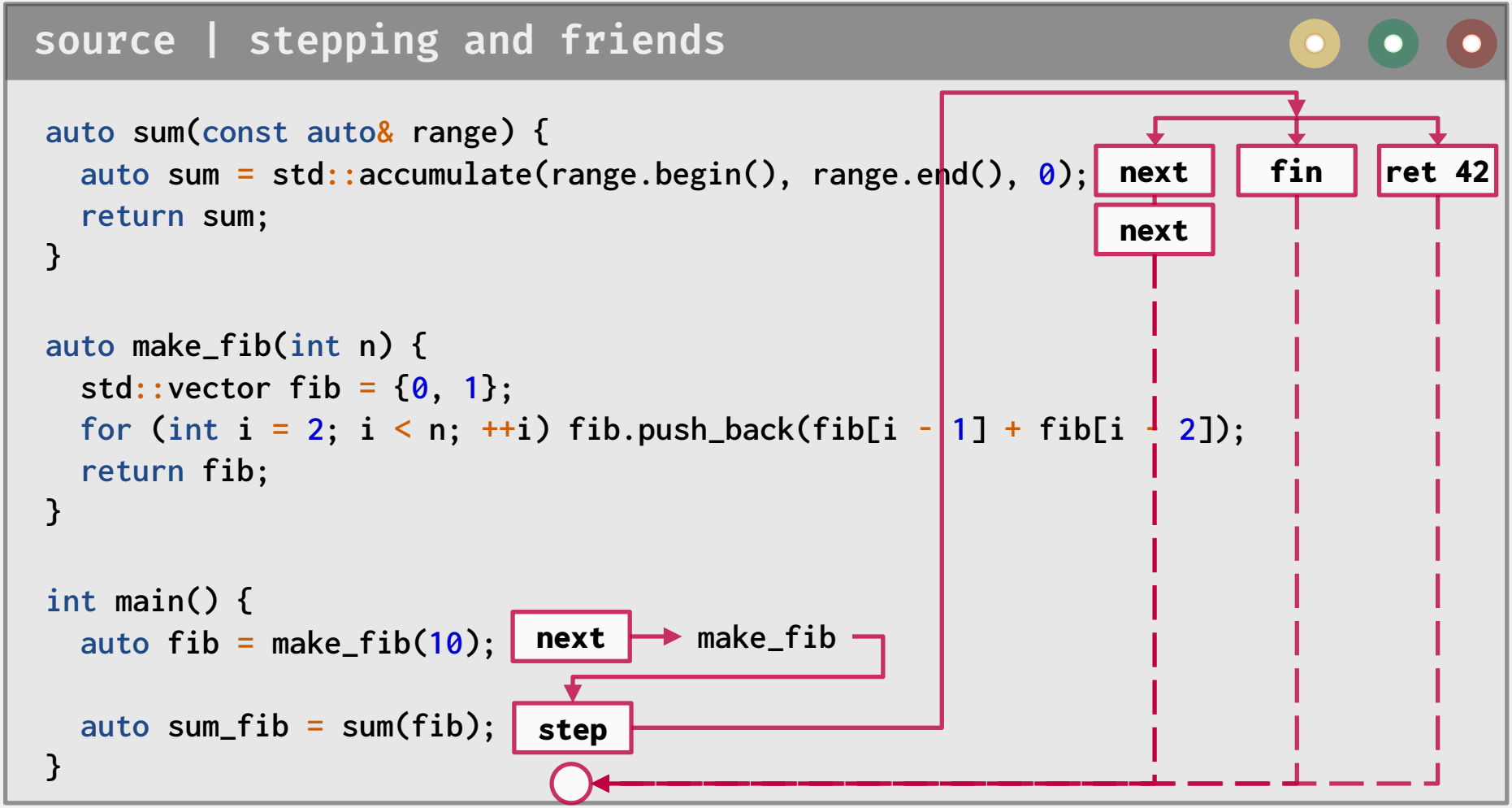
Command	What does it do
run or r	Starts executing the program
start	Starts executing the program and stops at the entry point
next or n	Continues execution until next source line
print or p<expr>	Evaluates expression and prints the result
list or l <what>	Lists the source code

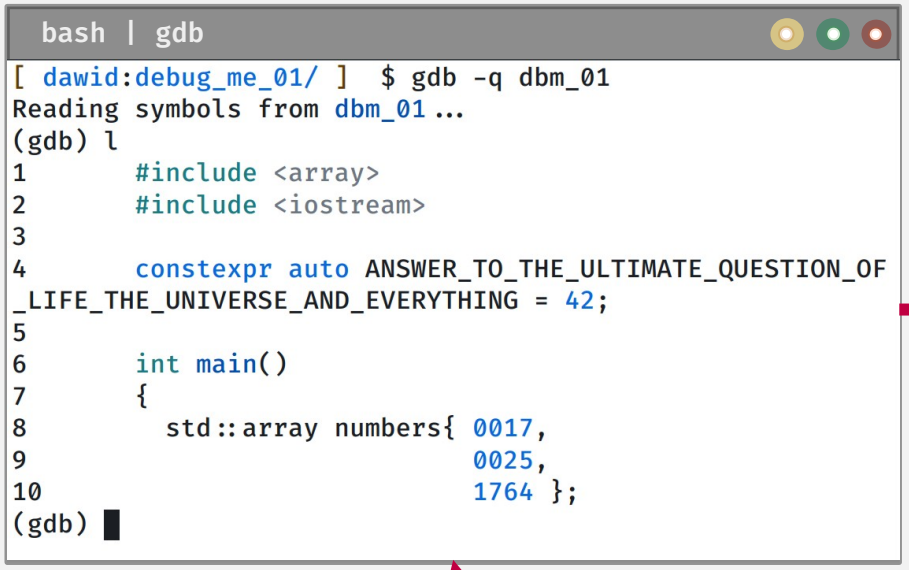
debug_me_01.cpp

- start & run
- next
- print
- list

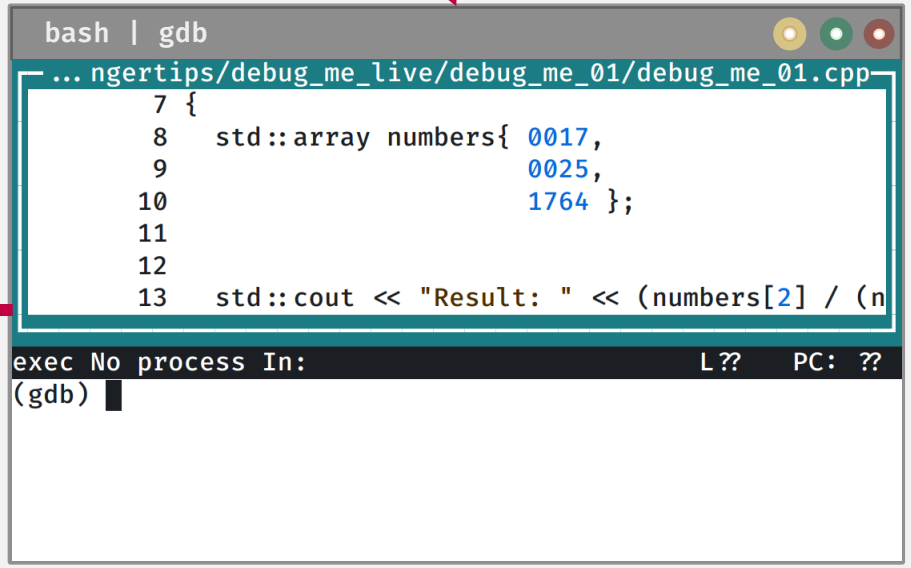
text | looking at source

Command	What code does it list
l file.cpp:line_number	Around line_number of file.cpp
l file.cpp:from,to	From line to line of file.cpp
l function_name	Around function function_name
l	A bit further from the previous list
l -	A bit back from the the previous list
l .	Around the current execution point
l *$pc	Around the current execution point (old gdb)


source | stepping and friends
auto sum(const auto& range) {
auto sum = std::accumulate(range.begin(), range.end(), 0);
return sum;
}
auto make_fib(int n) {
std::vector fib = {0, 1};
for (int i = 2; i < n; ++i) fib.push_back(fib[i - 1] + fib[i - 2]);
return fib;
}
int main() {
auto fib = make_fib(10);
auto sum_fib = sum(fib);
}
next
make_fib
step
next
next
fin
ret 42



Ctrl + x , a



Ctrl + x , a

Refresh: Ctrl + L

debug_me_02.cpp

- break
- continue
- next & step
- print
- finish
- return

text | stepping and friends

Command	What does it do
next or n	Continues execution to the next line in frame
step or s	Continues execution to the next line
continue or c	Continues execution to the next break
until <loc> or u <loc>	Continues execution until location
finish or fin	Continues execution to the end of function
ret <expr>	Returns the value of expr

source | frames

```
std::optional<result_type>
FibonacciGen::get_cached(std::size_t n)
{
  /*...*/
}

result_type FibonacciGen::generate(char n)
{
  auto cached_value = get_cached(n);
  /* ... */
}

int main() {
  FibonacciGen fib_gen{};
b auto Fn = fib_gen.generate(n);
}
```

step

frame #0

args: ---
locals: fib_gen, n, Fn

source | frames

```
std::optional<result_type>
FibonacciGen::get_cached(std::size_t n)
{
  /*...*/
}

result_type FibonacciGen::generate(char n)
{
  auto cached_value = get_cached(n);
  /* ... */
}

int main() {
  FibonacciGen fib_gen{};
b auto Fn = fib_gen.generate(n);
}
```

step

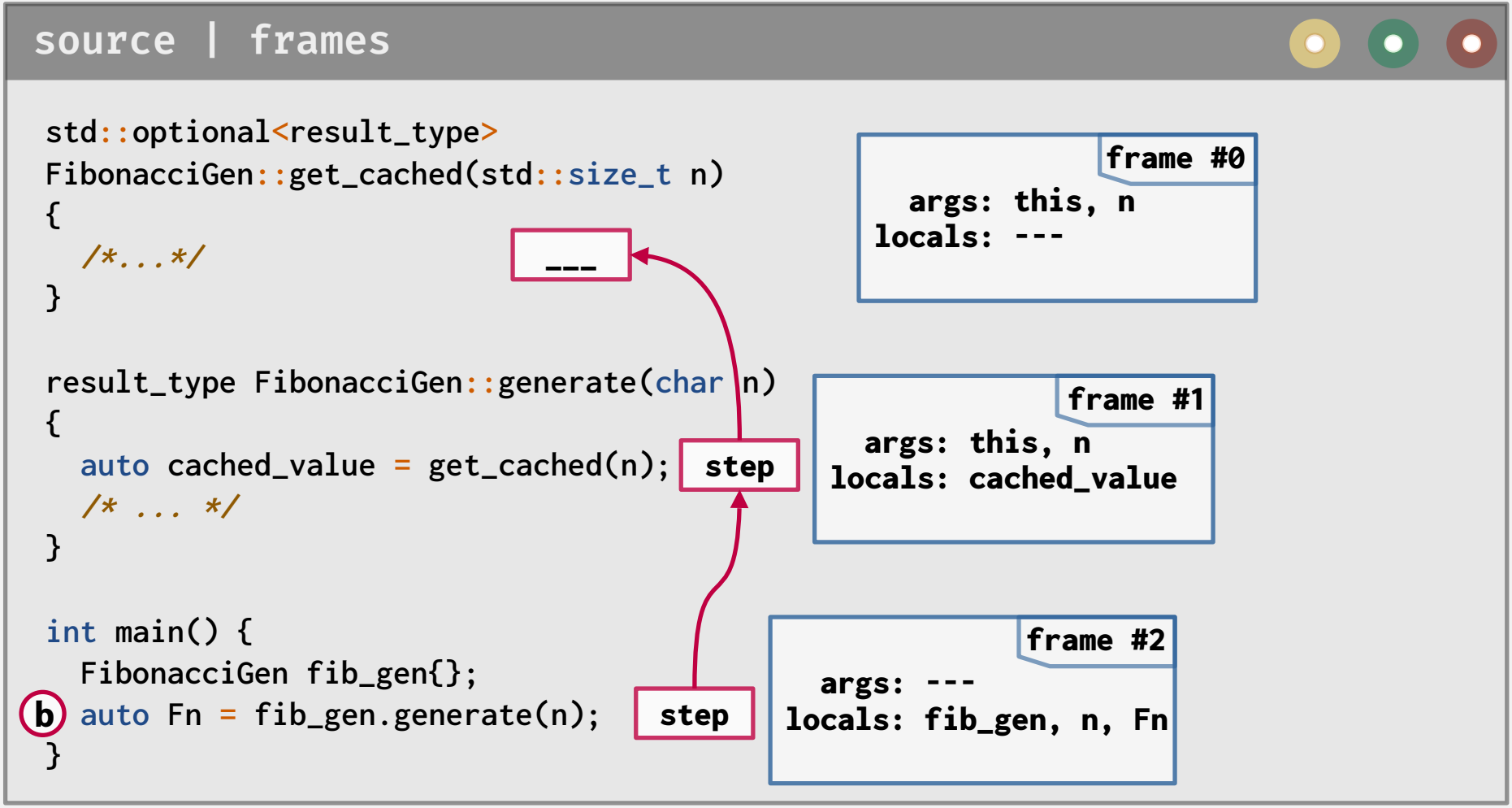
step

frame #0

args: this, n
locals: cached_value

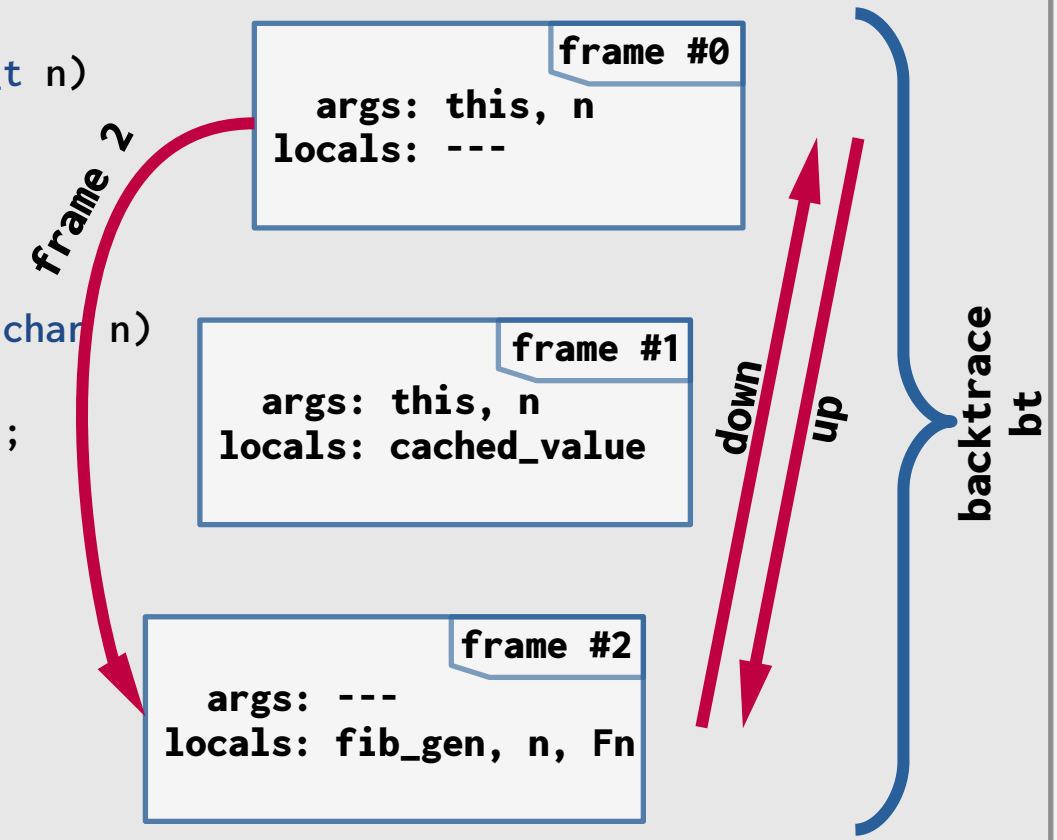
frame #1

args: ---
locals: fib_gen, n, Fn


source | frames
std::optional<result_type>
FibonacciGen::get_cached(std::size_t n)
{
/*...*/
}
result_type FibonacciGen::generate(char n)
{
auto cached_value = get_cached(n);
/* ... */
}
int main() {
FibonacciGen fib_gen{};
b auto Fn = fib_gen.generate(n);
}
___
step
step
frame #0
args: this, n
locals: ---
frame #1
args: this, n
locals: cached_value
frame #2
args: ---
locals: fib_gen, n, Fn

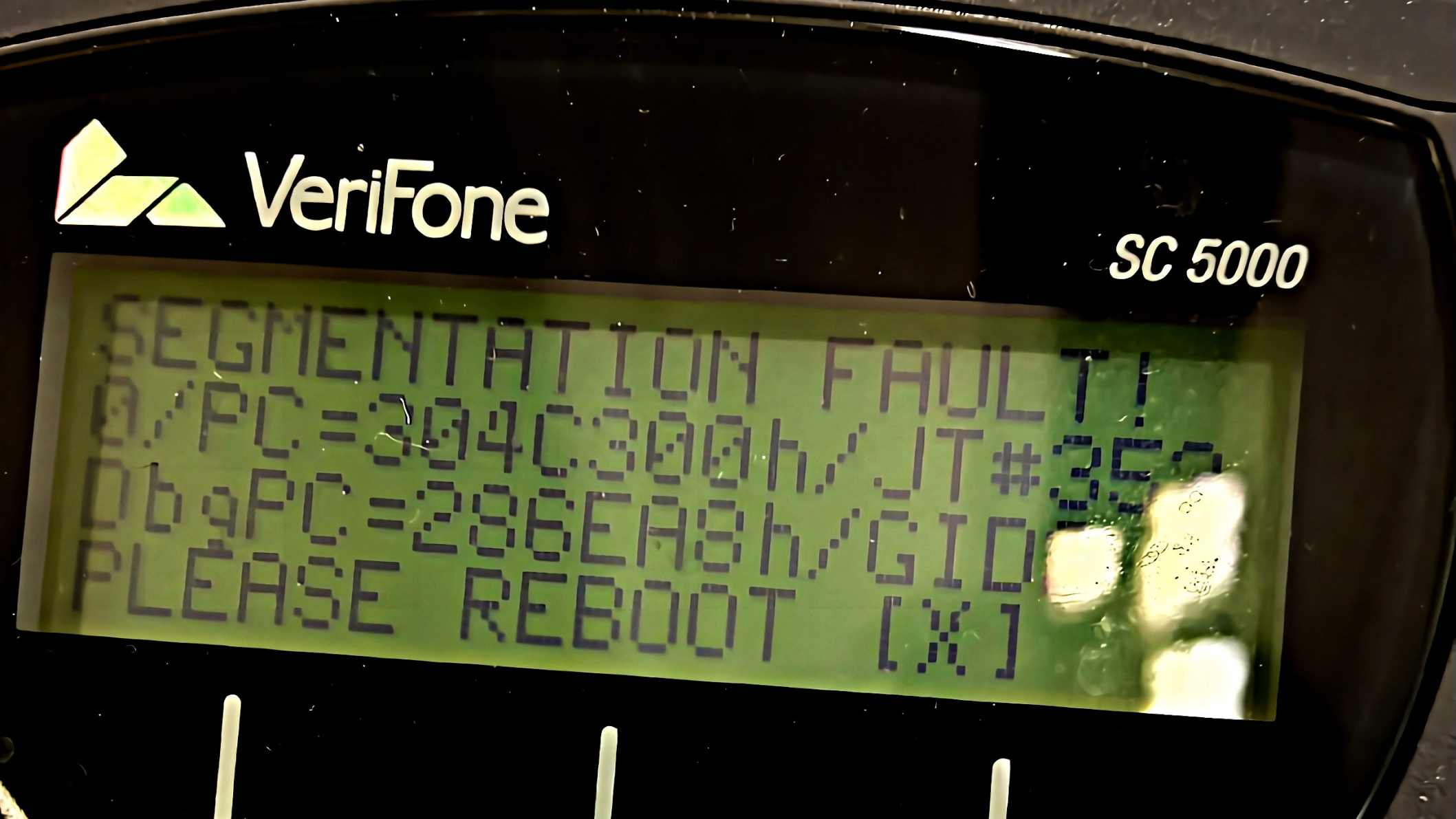
source | frames

```
std::optional<result_type>
FibonacciGen::get_cached(std::size_t n)
{
  /*...*/
}

result_type FibonacciGen::generate(char n)
{
  auto cached_value = get_cached(n);
  /* ... */
}

int main() {
  FibonacciGen fib_gen{};
  auto Fn = fib_gen.generate(n);
}
```



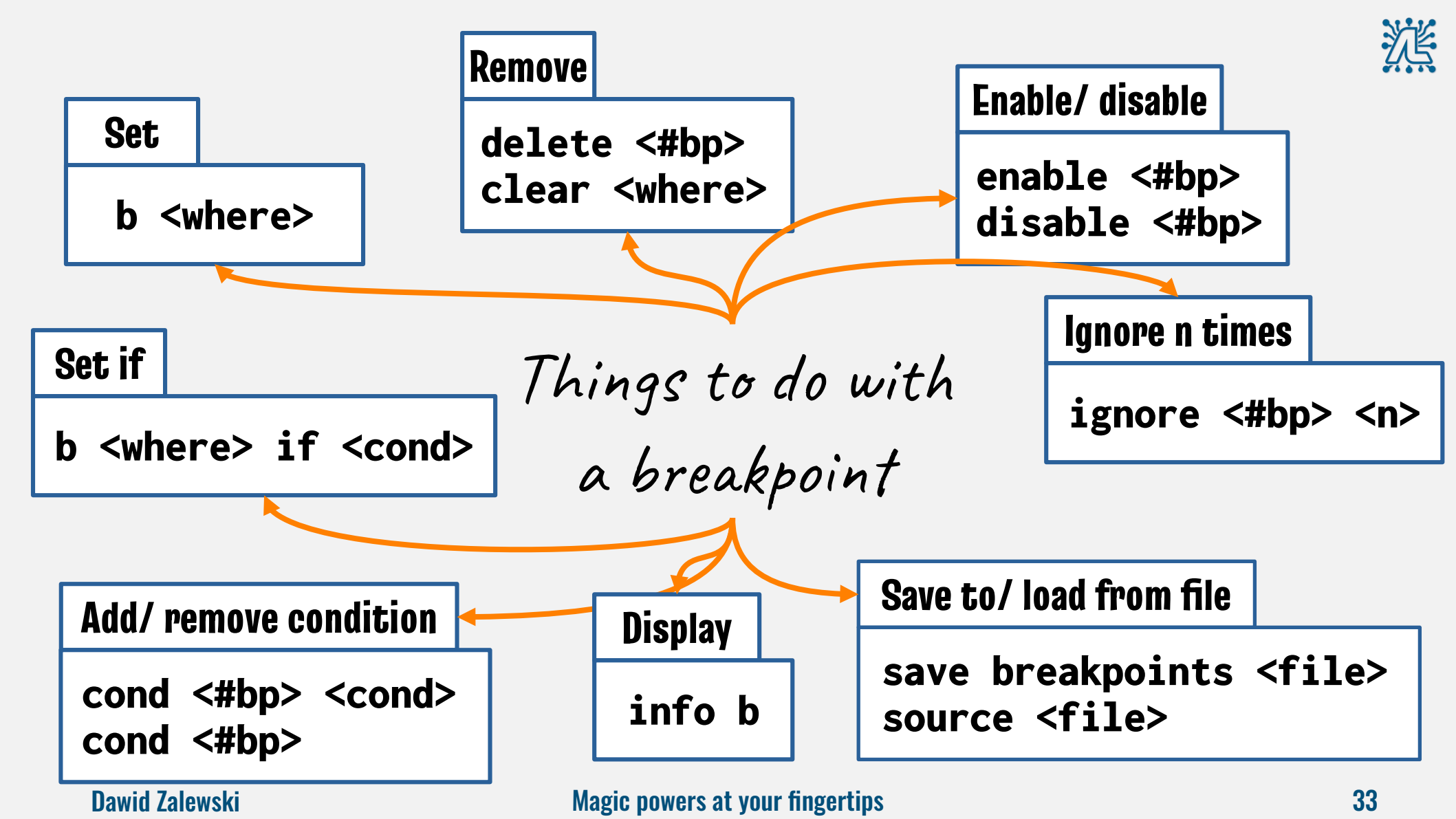
text | frames

Command	What does it do
backtrace or bt	Prints the backtrace (callstack)
bt full	Same with more details
frame or f	Prints frame information
f #num	Goes to frame #num
up #count	Goes #count frames up (default = 1)
down #count	Goes #count frames down (default = 1)
info args	Displays current function arguments
info locals	Disables local variables


VeriFone
SC 5000
SEGMENTATION FAULT!
0/PC=304C300h/JT#350
0b9PC=286EA8h/GIO
PLEASE REBOOT [X]

debug_me_03.cpp

- backtrace
- frame
- info locals
- info args


Set
b <where>
Remove
delete <#bp>
clear <where>
Enable/ disable
enable <#bp>
disable <#bp>
Ignore n times
ignore <#bp> <n>
Set if
b <where> if <cond>
Things to do with a breakpoint
Add/ remove condition
cond <#bp> <cond>
cond <#bp>
Display
info b
Save to/ load from file
save breakpoints <file>
source <file>

debug_me_03.cpp

- `bt full`
- `frame`
- `catch throw`

```
MoneyPot
- goal_     // “ Trip to the Moon”
- target_   // 1500
- deposits_
- deposits_size_
- deposits_capacity_

+ total()
+ add_deposit(...)
+ is_contributor(...)
+ total_for(...)
```

Deposit


A CORE DUMP

debug_me_04.cpp

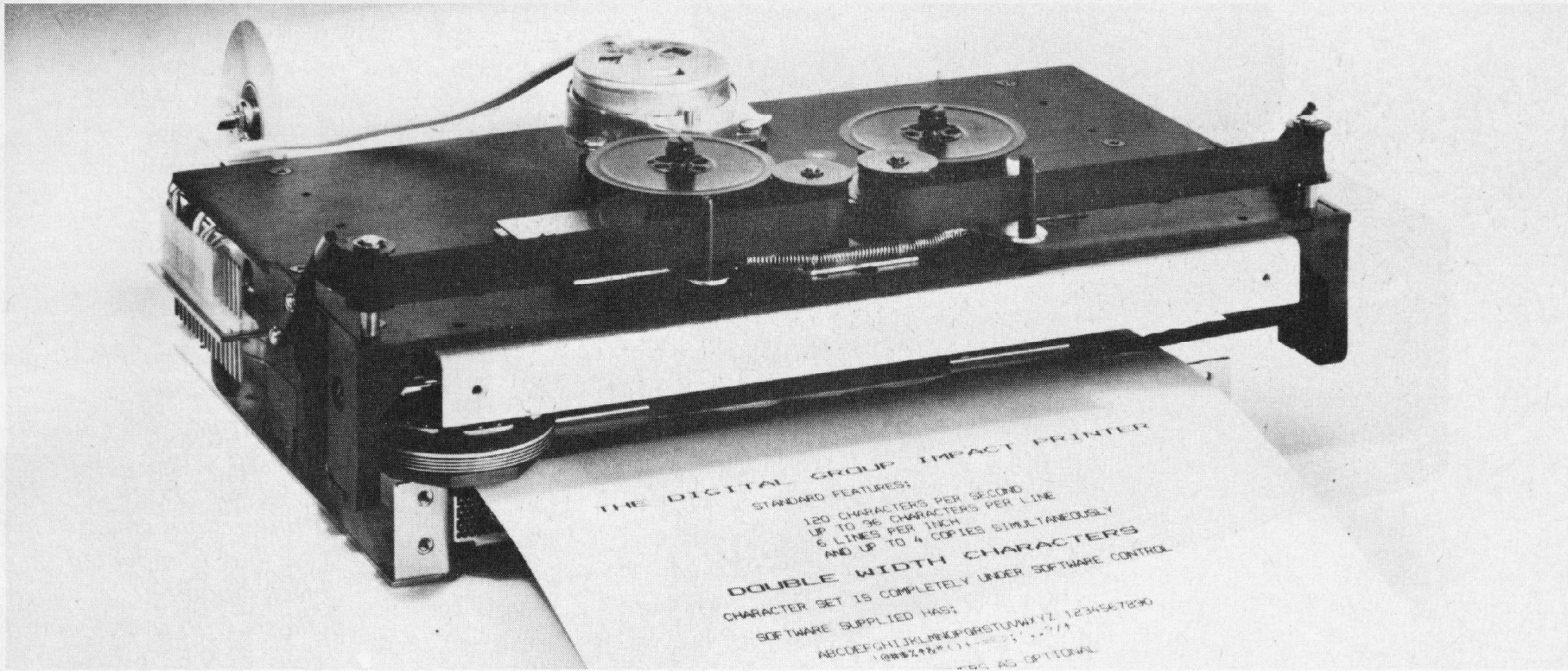
- core dump?

text | segfaulting with a dump

```
$ ./dbm_04
Segmentation fault (core dumped)

# ^^^ <- it's a lie

$ ulimit -c
0

$ ulimit -c unlimited
$ ./dbm_04
Segmentation fault (core dumped)

# ^^^ <- now we really have a core dump
```

debug_me_04.cpp

- core
- frame
- backtrace



Print Your Heart Out.

With help from the Digital Group, naturally.

text | displaying information

Command	What does it do
print <expr>	Evaluates expression and prints the reslult
disp <expr>	Prints value of <expr> when program stops
info disp	Shows active displays
undisp <#disp>	Deletes a display with number
dprintf <loc>, <fmt>, ...	Debug print

debug_me_04.cpp

- `dprinf`

source | debug print

```
============================== pot.cpp ================================

void Pot::add_deposit(std::string_view name, double amount)
{
  printf("Size: %d\n", deposits_size_);
  if (deposits_size_ == deposits_capacity_)
  /* ... */
}

================================ gdb ==================================

(gdb) dprintf, add_deposit, "Size: %d\n", deposits_size
```

source | debug print with %V

```
============================== pot.cpp ===============================

void Pot::add_deposit(std::string_view name, double amount)
{
  printf("Deposits: ");
  for (int i=0; i<deposits_size_; ++i) magically_print(deposits_[i]);
  printf("\n");
  /* ... */
}
================================ gdb =================================

(gdb) dprintf, add_deposit, "Deposits: %V\n", *deposits_@deposits_size_
```

debug_me_04.cpp

- display
- undisplay
- info display

text | modifying program's behavior

Command	What does it do
call <expr>	Evaluates exression
set var some_var=<expr>	Sets the value of a program's variable
$n	Recalls n-th result
$	Recalls last result
$$	Recalls one before last result
$$n	Recalls n before last result
set $my_var = <expr>	Creates internal gdb variable

debug_me_04.cpp

- call
- set var name = ...
- set $var = ...
- show convenience

Skip if needed

gdb | object creation

```
(gdb) call (int)printf("Hello, gdb!")
$1 = 10

(gdb) set $str = std::string("Hello, gdb!")
A syntax error in expression, near `("Hello, gdb!")'.

(gdb) set $str = 'std::string'("Hello, gdb!")
A syntax error in expression, near `("Hello, gdb!")'.
```

gdb | object creation

```
(gdb) set $pstr = (std::string*)malloc(sizeof(std::string))
(gdb) call $pstr->basic_string()
(gdb) call $pstr->operator=("Hello, gdb!)
$1 = "Hello, gdb!"
(gdb) p $pstr
$2 = (std::string *) 0x555555571800
(gdb) p *$pstr
$3 = "Hello, gdb!"

(gdb) call (int)puts($pstr->c_str())
$4 = "Hello, gdb!"
```

gdb | object creation

```
(gdb) set $psv = (std::string_view*)malloc(sizeof(std::string_view))
(gdb) call $psv->basic_string_view("HeLlO, wOrLd!")
(gdb) set $sv= *$psv
(gdb) p $sv
$1 = "HeLlO, wOrLd!"

(gdb) print 'pot::to_lower'($sv)
```

gdb | object creation

```
(gdb) call (void*) malloc(sizeof(std::string_view))
$1 = (void *) 0x555555571590

(gdb) call ((std::string_view*)$1)->basic_string_view("Hello world")
(gdb) print 'pot::to_lower'(*((std::string_view*)$1))
```

debug_me_05.cpp

- SIGABRT

gdb | passing arguments to programs

```
$ ./dmb_05 file.txt

$ gdb --args dbm_05 file.txt
(gdb) r

$ gdb dbm_05
(gdb) start file.txt

(gdb) show args
Argument list to give program being debugged when it is started
is "file.txt".

(gdb) set args
(gdb) set args input.txt
```

debug_me_05.cpp

- set args
- show args

3 theories of TIME TRAVEL

CREATED BY HARRISON DENSMORE

FIXED TIMELINE

IN A FIXED **TIMELINE**

EVEN WHEN PARTIES TRAVEL BACK IN TIME...

DYNAMIC TIMELINE

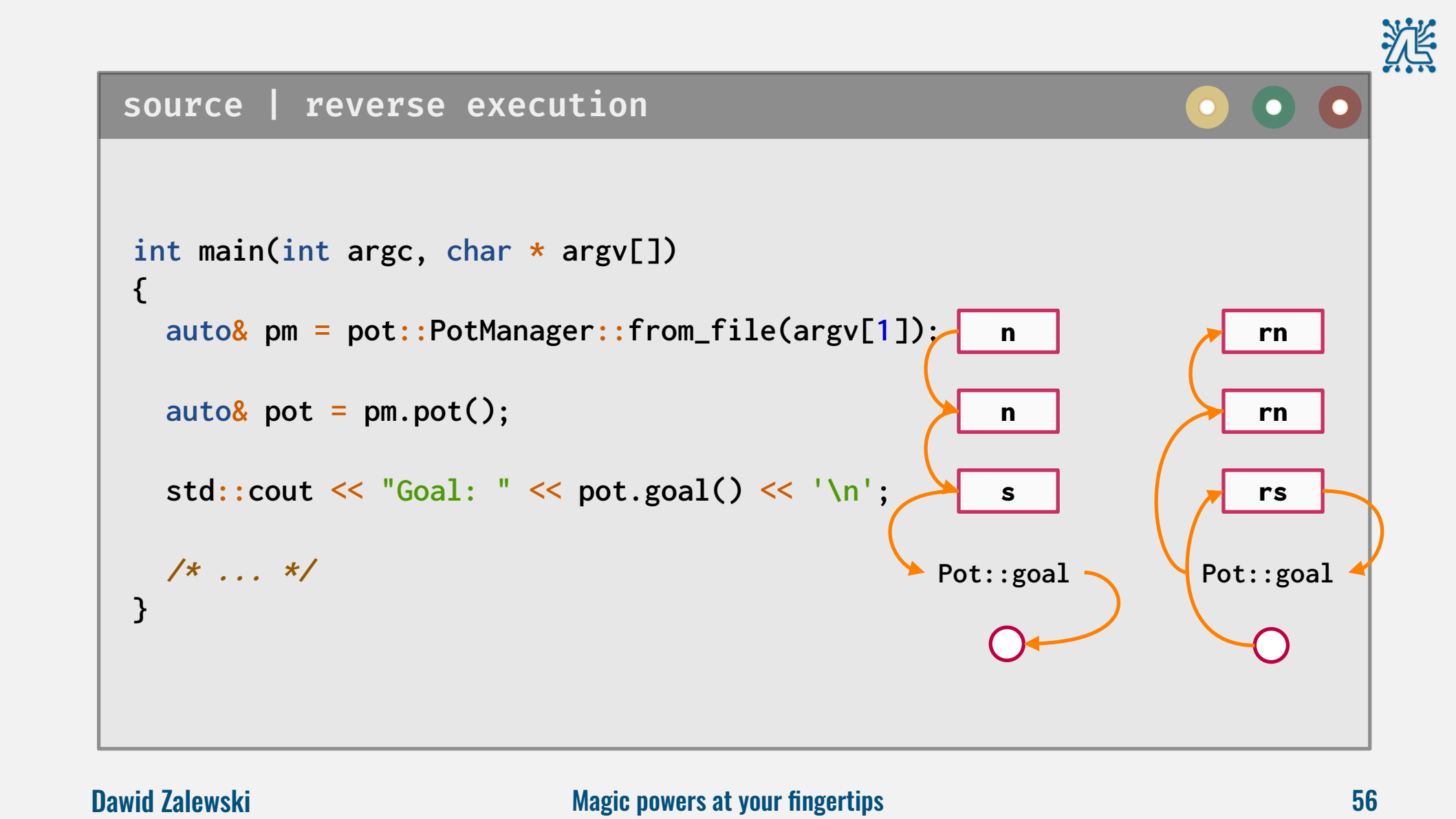
IN A DYNAMIC **TIMELINE**

ALTERED EVENTS IN THE PAST HAVE DEFINITE IMPACTS ON

MULTIVERSE

THE CONCEPT OF A MULTIVERSE SUPPORTS **ALTERNATE TIMELINES**

IN AN INFINITE NUMBER OF PARALLEL UNIVERSES.



source | reverse execution

```
int main(int argc, char * argv[])
{
  auto& pm = pot::PotManager::from_file(argv[1]);

  auto& pot = pm.pot();

  std::cout << "Goal: " << pot.goal() << '\n';

  /* ... */
}
```

gdb | reverse execution & recording

```
$ gdb -q ./my_app
(gdb) b crashing_function
Breakpoint 1 at ...
(gdb) r
Thread 1 "my_app" hit Breakpoint 1
(gdb) record full
(gdb) c
Thread 2 "dbm_06" received signal SIGSEGV, Segmentation fault.

(gdb) reverse-step
```

debug_me_05.cpp

- record (full)
- reverse-next
- reverse-step
- reverse-continue

gdb | disabling glibc features

```
$ gdb -q ./my_app

$ GLIBC_TUNABLES=glibc.cpu.hwcaps=-AVX,-AVX2,-AVX512BW,-AVX512DQ,
-AVX512F,-AVX512VL,-BMI1,-BMI2,-LZCNT,-MOVBE,-RTM,-SSE4_1,-SSE4_2,-SSSE3
gdb -q ./my_app
```

debug_me_05.cpp

- record (full)
- reverse-next
- reverse-step
- reverse-continue

text | reverse execution & recording

Command	What does it do
record	Starts recording program's execution
record stop	Stops recording
reverse-next	Next, but backward
rn	
reverse-step	Step into, but backward
rs	
reverse-continue	Continues, but backward
rc	
reverse-finish	Continues until the beginning of frame
reverse-fin	

rr | just rr it!

1. Visit: https://rr-project.org

2. Check if rr will work:

```
$ perf stat -e br_inst_retired.conditional true
$ # or on newer systems:
$ perf stat -e br_inst_retired.cond true
```

3. Most likely you'll need:

```
$ sudo sysctl kernel.perf_event_paranoid=1
```

4. Time-travel!

```
$ rr record ./my.app <args>
$ rr replay
(rr) b main
(rr) r
```

debug_me_05.cpp

- `rr record`
- `rr replay`

text | breakpoints

Command	What does it do
b file.cpp:line_number	Sets a breakpoint in file at line
b function_name	Sets a breakpoint at function
b file.cpp:function_name	Sets a breakpoint in file at function
b <args> if cond	A bit back from the the previous list
tb <args>	Sets a temporary breakpoint
ignore bp_num how_many	Ignores hits of a breakpoint how_many times
info break	Displays all breakpoints, watchpoints, ...
disable/ enable bp_num	Disables/ enables a breakpoint with bp_num
delete bp_num	Deletes a breakpoint with bp_num
save breakpoints fname	Saves breakpoint information to a file
source fname	Loads previously saved breakpoitns

text | watchpoints

Command	What does it do
watch var_name	Sets a watchpoint on a variable
watch -l expr	Sets a watchpoint on a memory address
watch <args> if cond	Sets a conditional watchpoint
watch <args> thread id	Sets a watchpoint in thread with id
ignore wp_num how_many	Ignores hits of a watchpoint how_many times
info watch	Displays all watchpoints
disable/ enable wp_num	Disables/ enables a breakpoint with bp_num
delete wp_num	Deletes a breakpoint with bp_num
rwatch	Sets a read wathpoint
awatch	Sets a read/ write watchpoint

debug_me_06.cpp

- watch
- watch if
- watch -location

gdb | breakpoints with commands

```
============================== logger.cpp ==============================

  void log(std::string_view msg)
  {
    if (sink_.available()) sink_.enqueue( msg );
  }


================================ gdb ================================

  (gdb) tb log
  Temporary breakpoint 42 at 0x555...
  (gdb) commands
  Type commands for breakpoint(s) 42, one per line.
  End with a line saying just "end".
  >rwatch -l *sink_.messages_ if ($_caller_matches("process_async", 0))
  >c
  >end
```

debug_me_06.cpp

- commands

- Compiling for debugging
- Running, stepping, returning
- Breakpoints
- Backtrace, frames, args, locals
- Print, display, dprintf
- Call, set, convenience variables
- Record, reverse-*, rr
- Watchpoints, commands, conditions

bash | getting gdb

```
# Install libs and deps
$ sudo apt install build-essential texinfo libmpfr-dev bison flex

$ wget https://sourceware.org/pub/gdb/releases/gdb-14.2.tar.gz

$ tar -xvzf gdb-14.2.tar.gz
$ cd gdb-14.2

$ ./configure -with-python=/usr/bin/python3
$ make -j8
$ sudo make install
```

bash | getting gdb

```
# Clone the gcc repo
$ git clone https://gcc.gnu.org/git/gcc.git

# Verify that the dir exists

$ test -d <path_to_cloned_repo>/libstdc++/python/
$ echo $?

# Enable printers and xfunctions in ~/.gdbinit
```

cat ~/.gdbinit

```
$ cat ~/.gdbinit

python
import sys
sys.path.insert(0, '<path_to_gcc_repo>/libstdc++-v3/python/')
from libstdcxx.v6.printers import register_libstdcxx_printers
register_libstdcxx_printers (None)
from libstdcxx.v6.xmethods import register_libstdcxx_xmethods
register_libstdcxx_xmethods(None)
end
```

```
GNU gdb (GDB) 14.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
```

```
commands
silent
watch -l data_ if $_any_caller_matches(
c
end
```

```
(gdb) dprintf Queue::pop "%d" head_
```

Thank you!

```
(gdb) set $pstr = (std::string *)malloc(sizeof(std::str
(gdb) call $pstr->basic_string()
(gdb) call $pstr->operator=("Hello, gdb!)
```

```
(gdb) r
Starting program: main.a
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Status] : pUUU??!?L??
Program received signal SIGSEGV, Segmentation fault.
(gdb) bt full
```

```
$ gdb -ex "r" --ar
```