

CMPT 473
Software Testing, Reliability and Security

Debugging

Nick Sumner
wsumner@sfu.ca

Debugging

- We have discussed

Debugging

- We have discussed
 - Handling bugs during execution
 - Submitting effective bug reports
 - Bug triage and management

Debugging

- We have discussed
 - Handling bugs during execution
 - Submitting effective bug reports
 - Bug triage and management

But *debugging* can require significant time and effort

Debugging

- We have discussed
 - Handling bugs during execution
 - Submitting effective bug reports
 - Bug triage and management

But *debugging* can require significant time and effort

- Debugging involves 2 keys issues

Debugging

- We have discussed
 - Handling bugs during execution
 - Submitting effective bug reports
 - Bug triage and management

But *debugging* can require significant time and effort

- Debugging involves 2 keys issues
 - **Understanding** why a program misbehaves

Debugging

- We have discussed
 - Handling bugs during execution
 - Submitting effective bug reports
 - Bug triage and management

But *debugging* can require significant time and effort

- Debugging involves 2 keys issues
 - Understanding why a program misbehaves
 - **Correcting** the behavior

Debugging

- We have discussed
 - Handling bugs during execution
 - Submitting effective bug reports
 - Bug triage and management

But *debugging* can require significant time and effort

- Debugging involves 2 keys issues
 - Understanding why a program misbehaves
 - Correcting the behavior

Anecdotally, the people I see who are best at debugging are also the best programmers.

Antipatterns in debugging

- Blaming the computer immediately

Antipatterns in debugging

- Blaming the computer immediately
 - Even if the computer is at fault, you don't know

Antipatterns in debugging

- Blaming the computer immediately
 - Even if the computer is at fault, you don't know
- Random changes (shotgun debugging)

Antipatterns in debugging

- Blaming the computer immediately
 - Even if the computer is at fault, you don't know
- Random changes (shotgun debugging)
- Random search

Antipatterns in debugging

- Blaming the computer immediately
 - Even if the computer is at fault, you don't know
- Random changes (shotgun debugging)
- Random search
- **Stack Overflow**
 - “If all of your friends drove off a cliff...”

Antipatterns in debugging

- Blaming the computer immediately
 - Even if the computer is at fault, you don't know
- Random changes (shotgun debugging)
- Random search
- Stack Overflow
 - “If all of your friends drove off a cliff...”
- **Good debugging:**

Antipatterns in debugging

- Blaming the computer immediately
 - Even if the computer is at fault, you don't know
- Random changes (shotgun debugging)
- Random search
- Stack Overflow
 - “If all of your friends drove off a cliff...”
- **Good debugging:**
 - Is systematic

Antipatterns in debugging

- Blaming the computer immediately
 - Even if the computer is at fault, you don't know
- Random changes (shotgun debugging)
- Random search
- Stack Overflow
 - “If all of your friends drove off a cliff...”
- **Good debugging:**
 - Is systematic
 - Progressively converges on the source of misbehavior

Antipatterns in debugging

- Blaming the computer immediately
 - Even if the computer is at fault, you don't know
- Random changes (shotgun debugging)
- Random search
- Stack Overflow
 - “If all of your friends drove off a cliff...”
- Good debugging:
 - Is systematic
 - Progressively converges on the source of misbehavior

Good debugging involves *investigation*.

Understanding bugs is an investigation

- Start by foregoing assumptions

Understanding bugs is an investigation

- Start by foregoing assumptions
 - Your mental model of the code is incorrect

Understanding bugs is an investigation

- Start by foregoing assumptions
 - Your mental model of the code is incorrect
 - The things you believed to be true were not

Understanding bugs is an investigation

- Start by foregoing assumptions
 - Your mental model of the code is incorrect
 - The things you believed to be true were not
 - The comments may not be correct

Understanding bugs is an investigation

- Start by foregoing assumptions
 - Your mental model of the code is incorrect
 - The things you believed to be true were not
 - The comments may not be correct
- Reproduce the bug

Understanding bugs is an investigation

- Start by foregoing assumptions
 - Your mental model of the code is incorrect
 - The things you believed to be true were not
 - The comments may not be correct
- Reproduce the bug
- **Ask: Why did the code produce the wrong behavior?**
 - Read the code
 - Think of several possibilities

Understanding bugs is an investigation

- Start by foregoing assumptions
 - Your mental model of the code is incorrect
 - The things you believed to be true were not
 - The comments may not be correct
- Reproduce the bug
- **Ask: Why did the code produce the wrong behavior?**
 - Read the code
 - Think of several possibilities

How can you identify the possible causes?
Can you write code to help?

Understanding bugs is an investigation

- Start by foregoing assumptions
 - Your mental model of the code is incorrect
 - The things you believed to be true were not
 - The comments may not be correct
- Reproduce the bug
- **Ask: Why did the code produce the wrong behavior?**
 - Read the code
 - Think of several possibilities
 - Each is a hypothesis about the buggy behavior

Understanding bugs is an investigation

- Start by foregoing assumptions
 - Your mental model of the code is incorrect
 - The things you believed to be true were not
 - The comments may not be correct
- Reproduce the bug
- Ask: Why did the code produce the wrong behavior?
 - Read the code
 - Think of several possibilities
 - Each is a hypothesis about the buggy behavior
- Rank the hypotheses

Understanding bugs is an investigation

- Start by foregoing assumptions
 - Your mental model of the code is incorrect
 - The things you believed to be true were not
 - The comments may not be correct
- Reproduce the bug
- Ask: Why did the code produce the wrong behavior?
 - Read the code
 - Think of several possibilities
 - Each is a hypothesis about the buggy behavior
- Rank the hypotheses
 - How easy are they to eliminate?
 - How likely are they to cause the bug?

Understanding bugs is an investigation

- Start by foregoing assumptions
 - Your mental model of the code is incorrect
 - The things you believed to be true were not
 - The comments may not be correct
- Reproduce the bug
- Ask: Why did the code produce the wrong behavior?
 - Read the code
 - Think of several possibilities
 - Each is a hypothesis about the buggy behavior
- Rank the hypotheses
 - How easy are they to eliminate?
 - How likely are they to cause the bug?
- Try to disprove each hypothesis
 - Collect more information & update your list as you go

Understanding bugs is an investigation

- Start by foregoing assumptions
 - Your mental model of the code is incorrect
 - The things you believed to be true were not
 - The comments may not be correct

• R
• A

**This should sound very familiar.
Why?**

- Each is a hypothesis about the buggy behavior
- Rank the hypotheses
 - How easy are they to eliminate?
 - How likely are they to cause the bug?
- Try to disprove each hypothesis
 - Collect more information & update your list as you go

The scientific method

- Understanding bugs is a *scientific* investigation

Ask a question

The scientific method

- Understanding bugs is a *scientific* investigation

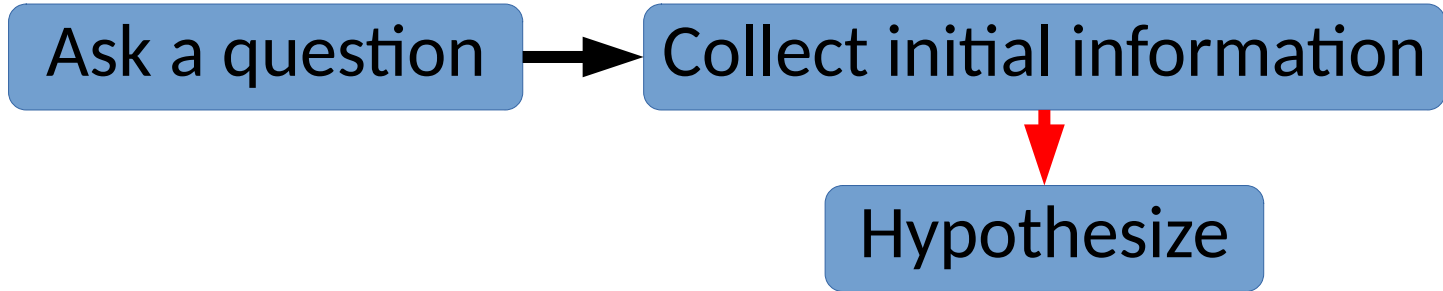
Ask a question



Collect initial information

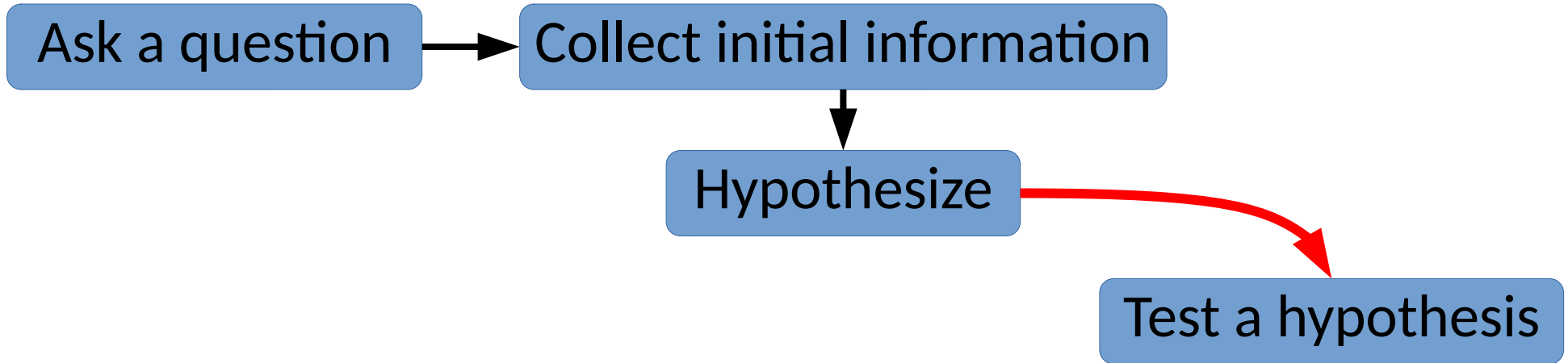
The scientific method

- Understanding bugs is a *scientific* investigation



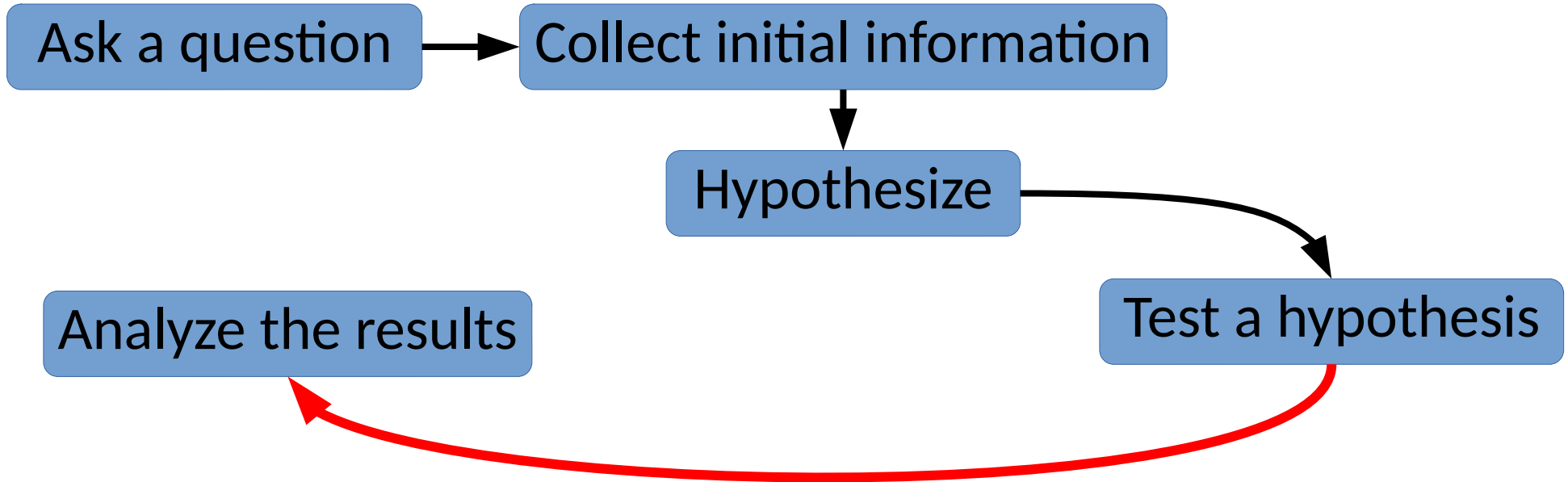
The scientific method

- Understanding bugs is a *scientific* investigation



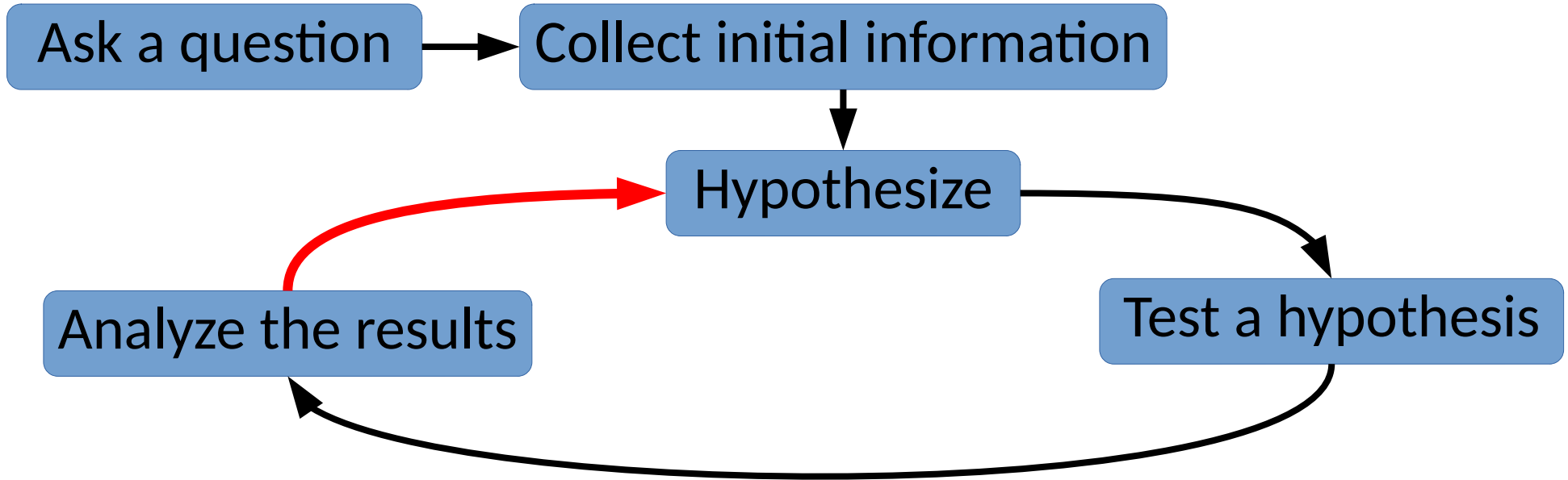
The scientific method

- Understanding bugs is a *scientific* investigation



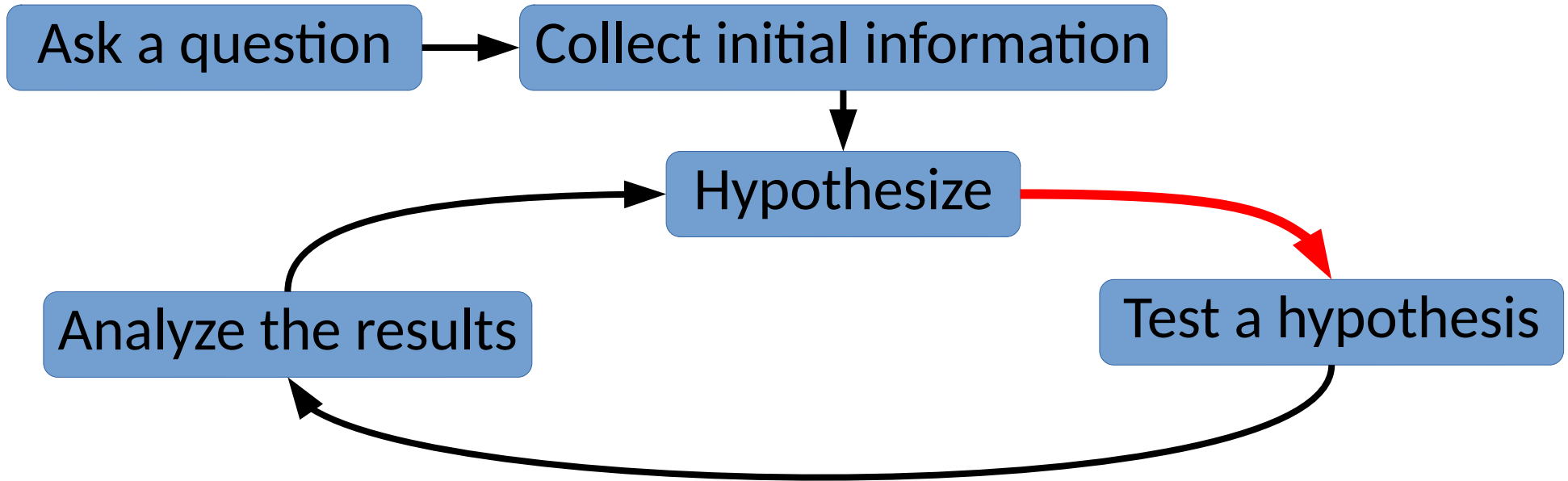
The scientific method

- Understanding bugs is a *scientific* investigation



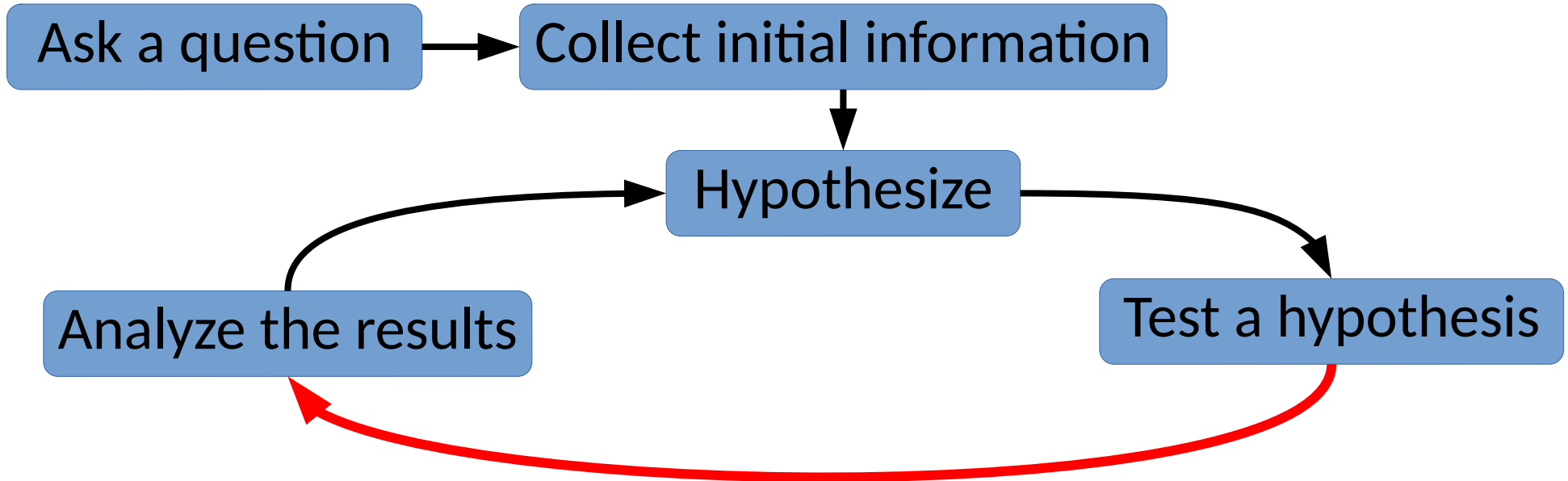
The scientific method

- Understanding bugs is a *scientific* investigation



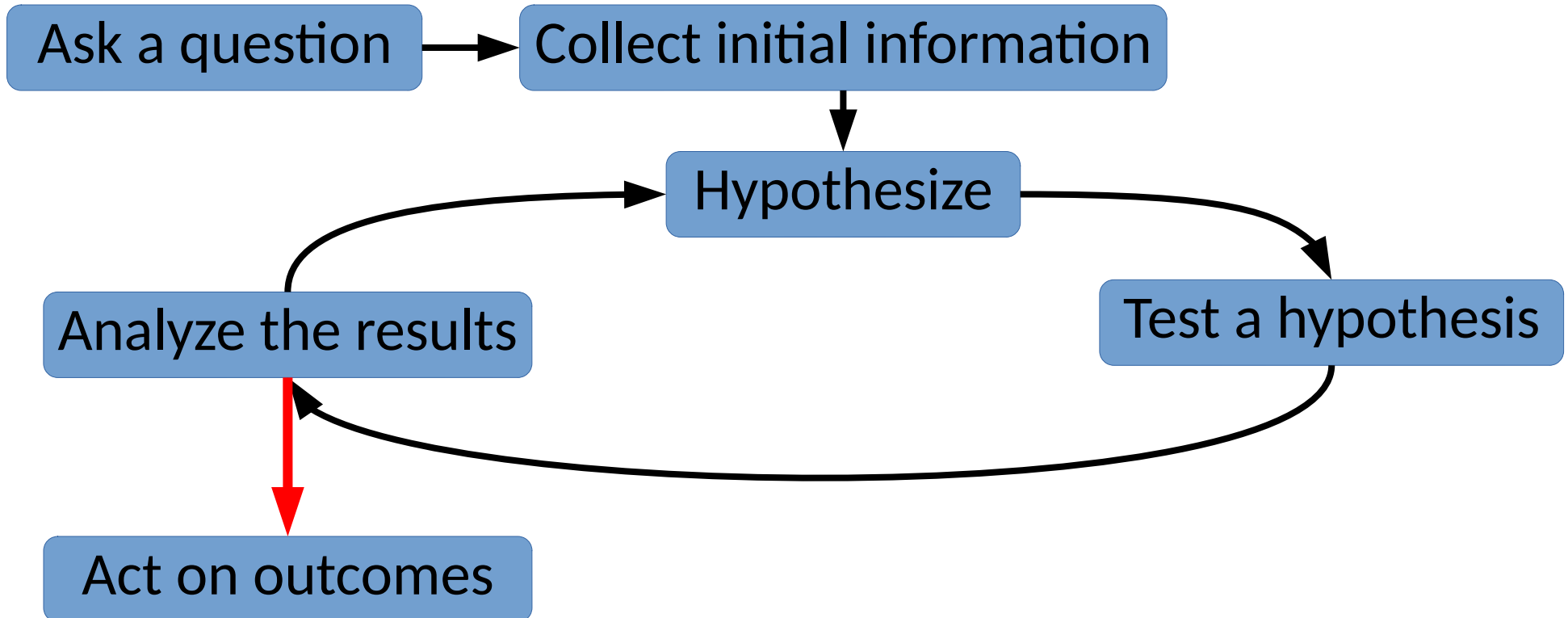
The scientific method

- Understanding bugs is a *scientific* investigation



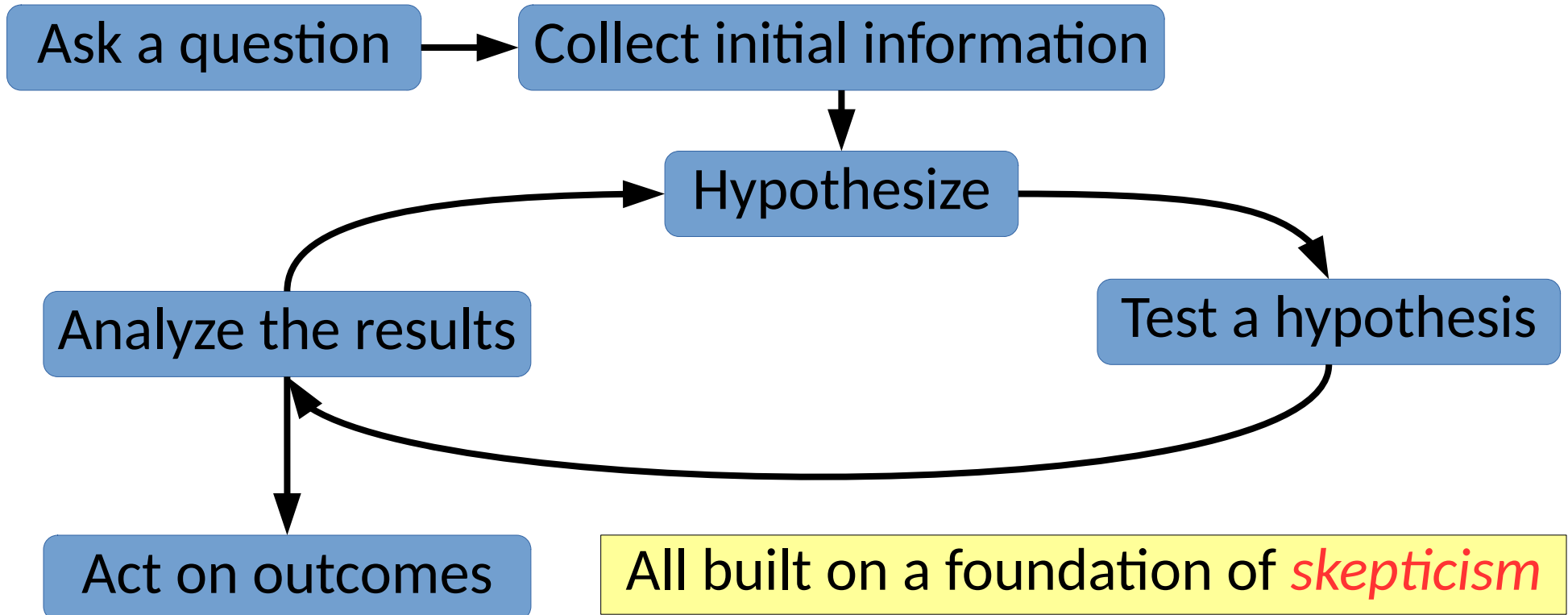
The scientific method

- Understanding bugs is a *scientific* investigation



The scientific method

- Understanding bugs is a *scientific* investigation



Debuggers

- Interactive debuggers are a key part of the investigation
 - Built into an IDE (like MSVC) or external (like GDB)

Debuggers

- Interactive debuggers are a key part of the investigation
 - Built into an IDE (like MSVC) or external (like GDB)
- Common set of features helps with
 - Fact finding
 - Identifying possible causes
 - Testing the causes as hypotheses

Debuggers

- Interactive debuggers are a key part of the investigation
 - Built into an IDE (like MSVC) or external (like GDB)
- Common set of features helps with
 - Fact finding
 - Identifying possible causes
 - Testing the causes as hypotheses
- **We will use GDB as a driving example**

Common Features

Basic commands for *exploring*

Common Features

Basic commands for *exploring*

- Running

```
gdb --args ./myprogram arg1 arg2  
...  
> run
```

Common Features

Basic commands for *exploring*

- Running

```
gdb --args ./myprogram arg1 arg2
...
> run
```

- Breakpoints

```
break meaningoflife.c:42
break foo
break foo if x > 0
```

Common Features

Basic commands for *exploring*

- Running

```
gdb --args ./myprogram arg1 arg2  
...  
> run
```

- Breakpoints

```
break meaningoflife.c:42  
break foo  
break foo if x > 0
```

- Stepping

```
step  
step 60  
next  
return
```

Common Features

Basic commands for *exploring*

- Running

```
gdb --args ./myprogram arg1 arg2  
...  
> run
```

- Breakpoints

```
break meaningoflife.c:42  
break foo  
break foo if x > 0
```

- Stepping

```
step  
step 60  
next  
return
```

- Continuing

```
continue  
finish
```

Common Features

Basic commands for *exploring*

- Running

```
gdb --args ./myprogram arg1 arg2
...
> run
```

- Breakpoints

```
break meaningoflife.c:42
break foo
break foo if x > 0
```

- Stepping

```
step
step 60
next
return
```

- Continuing

```
continue
finish
```

- Backtraces

```
bt
bt 5
bt -5
bt full 2
```


Common Features

Basic commands for *investigation*

Common Features

Basic commands for *investigation*

- Printing state

```
print x->y  
ptype x  
whatis x->foo()
```

Common Features

Basic commands for *investigation*

- Printing state
- Calling functions

```
print x->y  
ptype x  
whatis x->foo()
```

```
call foo()  
call printExtraInfo()  
call dumpData()
```

Common Features

Basic commands for *investigation*

- Printing state
- Calling functions
 - *Designing for debugging*

```
print x->y  
ptype x  
whatis x->foo()
```

```
call foo()  
call printExtraInfo()  
call dumpData()
```

Common Features

Basic commands for *investigation*

- Printing state

```
print x->y  
ptype x  
whatis x->foo()
```

- Calling functions
 - Designing for debugging

```
call foo()  
call printExtraInfo()  
call dumpData()
```

- Changing state and continuing (hypothesis testing)

```
set var x=42
```

Common Features

Basic commands for *investigation*

- Printing state

```
print x->y  
ptype x  
whatis x->foo()
```

- Calling functions
 - Designing for debugging

```
call foo()  
call printExtraInfo()  
call dumpData()
```

- Changing state and continuing (hypothesis testing)

```
set var x=42
```

- Watchpoints (breakpoints for data)

```
watch x
```

GDB Specifics

- TUI Mode

```
buggy.c
B+> 20     if (i == 42) {
21         ptrToX = (int*)0;
22     }
23     x += bar();
24 }

B+> 0x555555554705 <main+57>    cmpl  $0x2a,-0x18(%rbp)
0x555555554709 <main+61>    jne   0x555555554713 <main+71>
0x55555555470b <main+63>    movq  $0x0,-0x10(%rbp)
0x555555554713 <main+71>    mov   $0x0,%eax
0x555555554718 <main+76>    callq 0x5555555546aa <bar>
0x55555555471d <main+81>    mov   %eax,%edx

native process 12135 In: main          L20  PC: 0x555555554705
(gdb) run
Starting program: /home/nick/teaching/473/debugging/a.out

Breakpoint 1, main () at buggy.c:20
(gdb) █
```

GDB Specifics

- TUI Mode

The screenshot shows the GDB TUI interface for a program named 'buggy.c'. The top pane displays the source code with line numbers 20 through 24. The middle pane shows the assembly code for the current instruction. A blue overlay box in the foreground lists various GDB TUI commands and their shortcuts.

```
buggy.c
B+> 20     if (i == 42) {
21         ptrToX = (int*)0;
22     }
23     x += bar();
24 }
```

```
B+> 0x555555554705 <main+57>    cmpl   $0x2a,-0x18(%rbp)
0x555555554709 <main+61>    jne    0x555555554713 <main+71>
0x55555555470b <main+63>    movq   $0x0,-0x10(%rbp)
0x555555554713 <main+71>    mov    $0x0,%eax
0x555555554718 <main+76>    callq 0x5555555546aa <bar>
0x55555555
```

Enter	ctrl-x-a
Repaint	ctrl-l
Window Cycle	ctrl-x-2
"" in reverse	ctrl-x-1
Previous Command	ctrl-p
Next Command	ctrl-n

native process
(gdb) run
Starting program: /.../...
Breakpoint 1,
(gdb) █

GDB Specifics

- Built in Python interpreter
 - Defining your own GDB commands
 - Programmatic breakpoint manipulation

```
> python  
...
```



Reverse Execution

- Available in GDB, MSVC, Mozilla RR, ...

Reverse Execution

- Available in GDB, MSVC, Mozilla RR, ...
- Mozilla RR (record & replay based debugging)
 - Records behavior to a trace file
 - Allows deterministic replay of the same execution

Reverse Execution

- Available in GDB, MSVC, Mozilla RR, ...
- Mozilla RR (record & replay based debugging)
 - Records behavior to a trace file
 - Allows deterministic replay of the same execution
 - The trace may even be shared across computers

Reverse Execution

- Available in GDB, MSVC, Mozilla RR, ...
- **Mozilla RR (record & replay based debugging)**
 - Records behavior to a trace file
 - Allows deterministic replay of the same execution
 - The trace may even be shared across computers
 - System design enables ***running an execution backward***

Reverse Execution

- Available in GDB, MSVC, Mozilla RR, ...
- **Mozilla RR (record & replay based debugging)**
 - Records behavior to a trace file
 - Allows deterministic replay of the same execution
 - The trace may even be shared across computers
 - System design enables running an execution backward

```
rr record /path/to/my/program --args  
rr replay
```

Reverse Execution

`reverse-continue`
`reverse-step`
`reverse-next`
`reverse-finish`

Reverse Execution

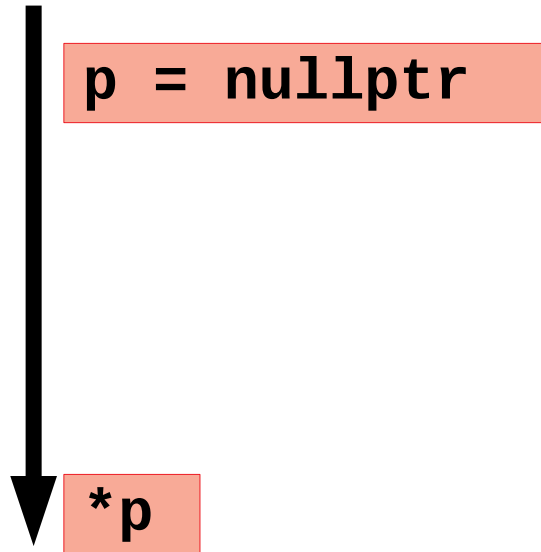
```
reverse-continue  
reverse-step  
reverse-next  
reverse-finish
```

- Interacting with watchpoints & breakpoints

Reverse Execution

```
reverse-continue  
reverse-step  
reverse-next  
reverse-finish
```

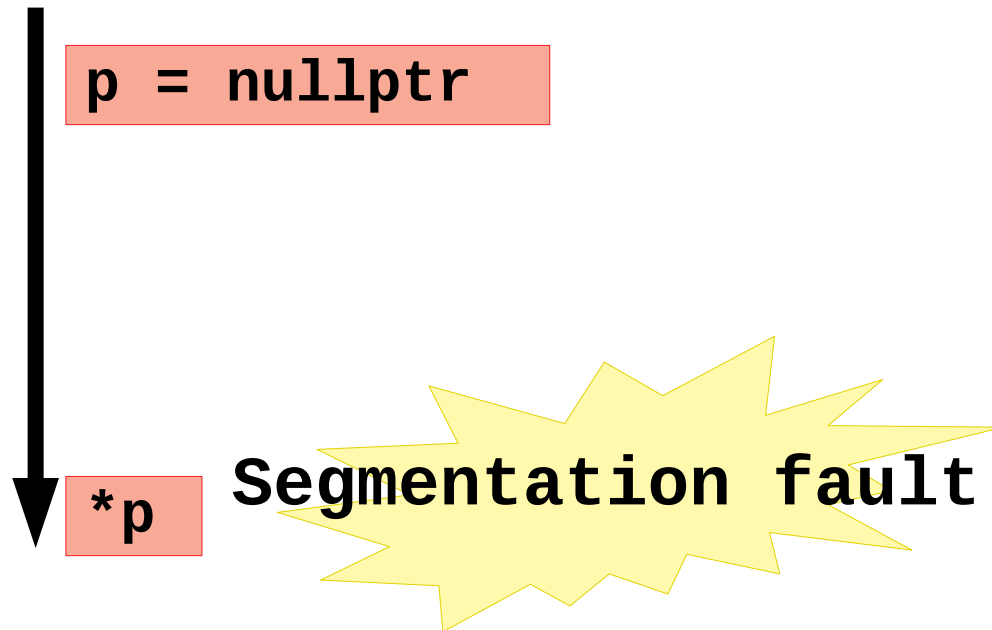
- Interacting with watchpoints & breakpoints



Reverse Execution

reverse-continue
reverse-step
reverse-next
reverse-finish

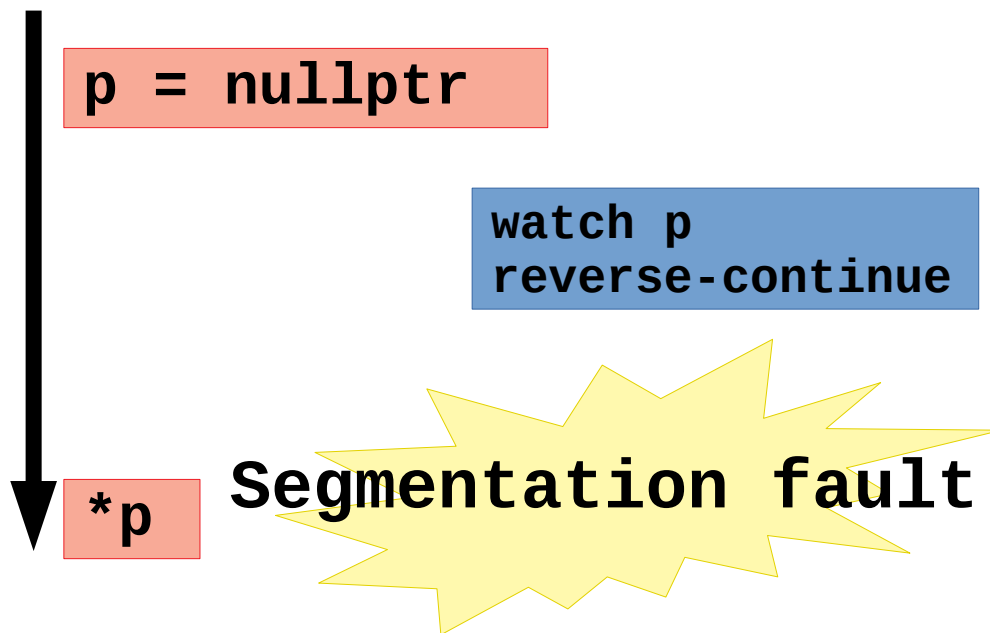
- Interacting with watchpoints & breakpoints



Reverse Execution

```
reverse-continue  
reverse-step  
reverse-next  
reverse-finish
```

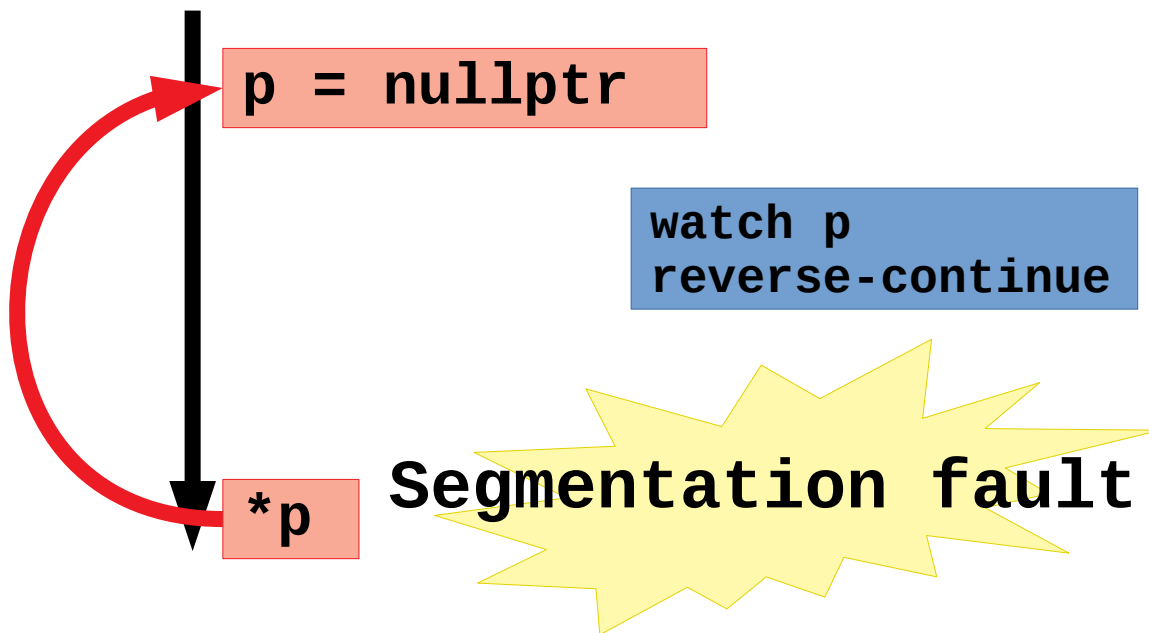
- Interacting with watchpoints & breakpoints



Reverse Execution

reverse-continue
reverse-step
reverse-next
reverse-finish

- Interacting with watchpoints & breakpoints



Summary

- Good debugging follows a methodical process

Summary

- Good debugging follows a methodical process
- Iteratively get closer to the buggy behavior

Summary

- Good debugging follows a methodical process
- Iteratively get closer to the buggy behavior
- **Make the most of your investigative tools**