Föreläsning 7

Lars-Henrik Eriksson

Föreläsningsbilderna är baserade på bilder gjorda av Tobias Wrigstad.

Testning CUnit

"Software bugs cost the U.S. economy
an estimated $59.5 billion per year.
An estimated $22.2 billion
could be eliminated by
improved testing that enables
earlier and more effective
identification and removal of defects."

- US Department of Commerce (NIST) 2002

Obs. att det är amerikanska biljoner, dvs en svensk miljard (10^9).
Kostnaden uppges ha ökat till över en svensk biljon (10^{12}) idag.

Finn felet!

```
int count_spaces(char *str)
{
  int length = strlen(str);
  int count = 0;
  for(int i = 1; i < length; ++i)
    {
      if(str[i] == ' ')
        count++;
    }
  return(count);
}
```

Finn felet!

```
int count_spaces(char *str)
{
  int length = strlen(str);
  int count = 0;
  for(int i = 1; i < length; ++i)
    {
      if(str[i] == ' ')
        count++;
    }
  return(count);
}
```

Bug: i = 1 istf i = 0 i for-loopen

Inte helt enkel att upptäcka, t.ex.
`str = "A B C"` ser inte felet, men
`str = " L"` ser det

Testning kan bara påvisa förekomsten av fel, inte avsaknaden

— Dijkstra

Ett test som inte kan utföras automatiskt är ett dåligt test!

Vad är ett test?

- Låt säga att vi vill testa `f(x) = y` för någon given ”funktion” `f`

  x är indata
  y är utdata
  x och y **tillsammans** är ett **test**, eller **testfall**

- Exempelvis, för `f = ++` (prefix)

```
++42 = 43          (x = 42, y = 43)
++0 = 1
++-1 = 0
```

Skall vi verkligen göra 2^{32} tester?

Varför har vi valt just dessa testfall?

x	y	Anmärkning
42	43	Vanlig aritmetik
0	1	Gräns, från noll till positiv
-1	0	Gräns, från negativ till noll

Testdesign

- Det är svårt att skriva bra tester
- Kräver kunskap om domänen
  - *Vilka är gränsfallen*
  - *Vilka är de vanliga felen*
- Vad vet vi om programmet?
  - *Black box testing*
  - *White box testing*
- Två syner på test
  - *Hur kan jag ha sönder programmet?*
  - *Hur kan jag förbättra programmet?*

Den här kursen nosar bara på testning

- **Enhetstestning** (dagens fokus)

  Pröva en liten del av programmet i isolation

- **Integrationstestning** (under projektet)

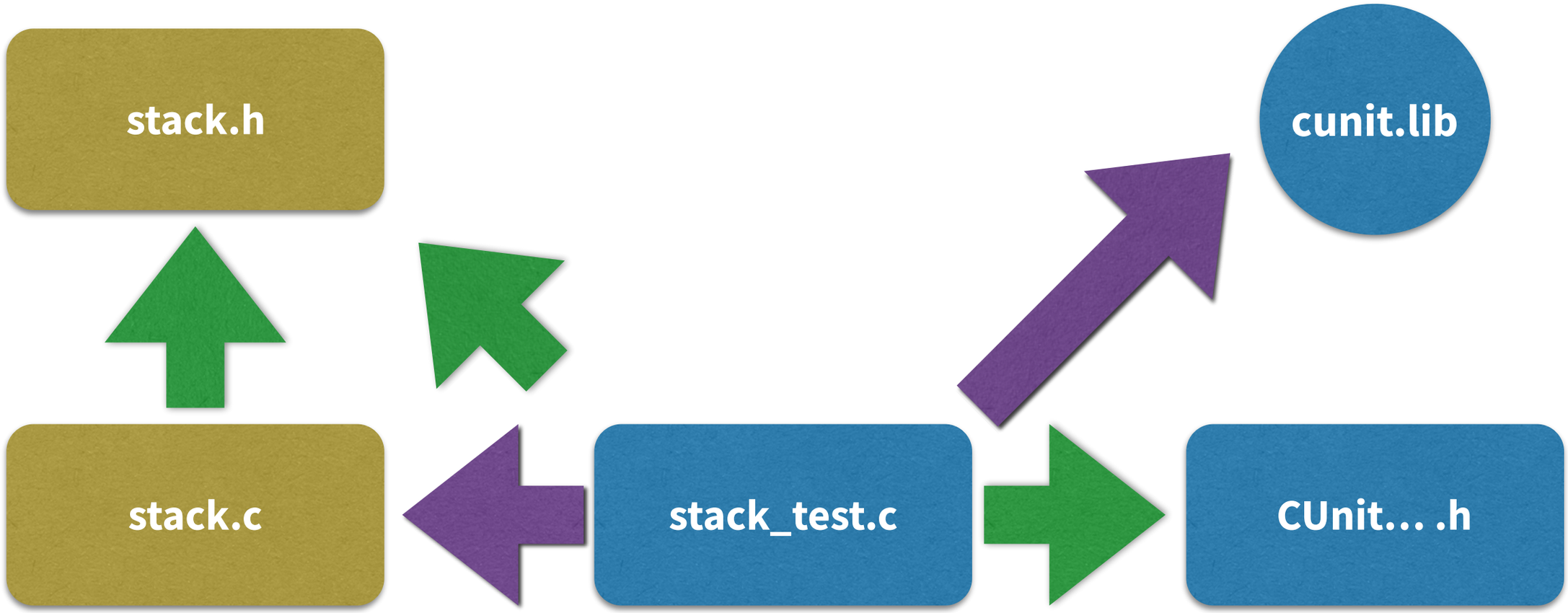
  Hur olika moduler/programdelar samverkar

- **Regressionstestning** (under projektet)

  Undvik regression vid underhåll/buggfixning

  (Regression = återgång till tidigare fel)

Det som skall testas *Testerna* *Testbibliotek*



Ett tests anatomi

- En test = en funktion
- Skapa data som behövs för testet
- Utför testet

  Operationer varvat med ASSERTs

- Riv ned testet

```
void test_stack_creation()
{
  stack_t *s = ioopm_stack_create(10);

  CU_ASSERT_FALSE(s == NULL);
  CU_ASSERT_TRUE(ioopm_stack_size(s) == 0);

  ioopm_stack_destroy(s);
}
```

- Viktigt att testa så litet som möjligt i varje enskilt test
- Viktigt att riva skapa och riva ned vid varje test

  Många verktyg (inkl. CUnit) har stöd för att göra det lättare

http://cunit.sourceforge.net/doc/writing_tests.html#assertions

Asserts (35 stycken!)

Assert	...som avbryter testfallet
CU_ASSERT_TRUE(value)	CU_ASSERT_TRUE_FATAL(value)
CU_ASSERT_FALSE(value)	CU_ASSERT_FALSE_FATAL(value)
CU_ASSERT_EQUAL(actual, expected)	CU_ASSERT_EQUAL_FATAL(actual, expected)
CU_ASSERT_NOT_EQUAL(actual, expected))	CU_ASSERT_NOT_EQUAL_FATAL(actual, expected))
CU_ASSERT_PTR_EQUAL(actual, expected)	CU_ASSERT_PTR_EQUAL_FATAL(actual, expected)
CU_ASSERT_PTR_NULL(value)	CU_ASSERT_PTR_NULL_FATAL(value)
CU_ASSERT_PTR_NOT_NULL(value)	CU_ASSERT_PTR_NOT_NULL_FATAL(value)
CU_ASSERT_STRING_EQUAL(actual, expected)	CU_ASSERT_STRING_EQUAL_FATAL(actual, expected)
CU_FAIL(message)	CU_FAIL_FATAL(message)

Exempel

```
void *data = malloc(2048);
CU_ASSERT_PTR_NOT_NULL(data);
```

```
char buf[256];
scanf("%s", buf);
char *word = my_copy_to_heap_function(buf);
CU_ASSERT_STRING_EQUAL(buf, word);
```

```
treenode_t *t = treenode_new_leaf(key, data);
CU_ASSERT_TRUE(t->key == key);
CU_ASSERT_TRUE(t->data == data);
CU_ASSERT_PTR_NULL(t->left);
CU_ASSERT_PTR_NULL(t->right);
```

inkapsling?

Exempel

```
void *data = malloc(2048);
CU_ASSERT_PTR_NOT_NULL(data);
```

```
char buf[256];
scanf("%s", buf);
char *word = my_copy_to_heap_function(buf);
CU_ASSERT_STRING_EQUAL(buf, word);
```

```
treenode_t *t = treenode_new_leaf(key, data);
CU_ASSERT_TRUE(tn_get_key(t) == key);
CU_ASSERT_TRUE(tn_get_data(t) == data);
CU_ASSERT_PTR_NULL(tn_get_left(t));
CU_ASSERT_PTR_NULL(tn_get_right(t));
```

vad testas?

Regressionstestning

- En (stor) uppsättning enhetstester för olika delar av programmet
- Möjlighet att köra dem automatiskt

- Vid varje förändring av programmet:

  Kör alla enhetstester automatiskt

  Notera vilka som passerar och vilka som inte passerar

  Implementera förändringen

  Kör alla enhetstester automatiskt igen

  Stäm av mot föregående lista — verkar det rimligt?

Table 20-2. Defect-Detection Rates

Removal Step	Lowest Rate	Modal Rate	Highest Rate
Informal design reviews	25%	35%	40%
Formal design inspections	45%	55%	65%
Informal code reviews	20%	25%	35%
Formal code inspections	45%	60%	70%
Modeling or prototyping	35%	65%	80%
Personal desk-checking of code	20%	40%	60%
Unit test	15%	30%	50%
New function (component) test	20%	30%	35%
Integration test	25%	35%	40%
Regression test	15%	25%	30%
System test	25%	40%	55%
Low-volume beta test (<10 sites)	25%	35%	40%
High-volume beta test (>1,000 sites)	60%	75%	85%

Från McConnell's Code Complete

Vad skall man testa för något?

stack.h

```
typedef struct stack stack_t;

struct stack
{
  int height;
  int maxsize;
  int values[0];
};

......
```

stack.c

```
#include <stdlib.h>
#include <stdbool.h>
#include "stack.h"

stack_t *ioopm_stack_create(int maxsize)
{
  stack_t *result =
            malloc(sizeof(stack_t) +
                   maxsize*sizeof(int));
  if (result)
    {
      *result = (stack_t){ .height = 0,
                           .maxsize = maxsize };
    }
  return result;
}

void ioopm_stack_destroy(stack_t *s)
{
  free(s);
}

int ioopm_stack_top(stack_t *s)
{
  return s->values[s->height-1];
}

int ioopm_stack_size(stack_t *s)
{
  return s->height;
}
```

```
bool ioopm_stack_push(stack_t *s, int v)
{
  if (s->height < s->maxsize)
    {
      s->values[s->height++] = v;
      return true;
    }
  else
    {
      return false;
    }
}

bool ioopm_stack_pop(stack_t *s)
{
  if (s->height > 0)
    {
      s->height--;
      return true;
    }
  else
    {
      return false;
    }
}
```

Vad är ett bra test?

Coverage — testtäckning

- För att testkvaliteten skall vara hög vill vi testa så många delar som möjligt av `f`
- För att minska kostnader vill vi undvika fler än ett test som testar samma sak

  Handlar också om omöjligheten att testa alla fall

- Vi skall se `gcov` senare under denna föreläsning

Strukturell testtäckning

```
bool ioopm_stack_pop(stack_t *s)
{
  if (s->height > 0)
    {
      s->height--;
      return true;
    }
  else
    {
      return = false;
    }
```

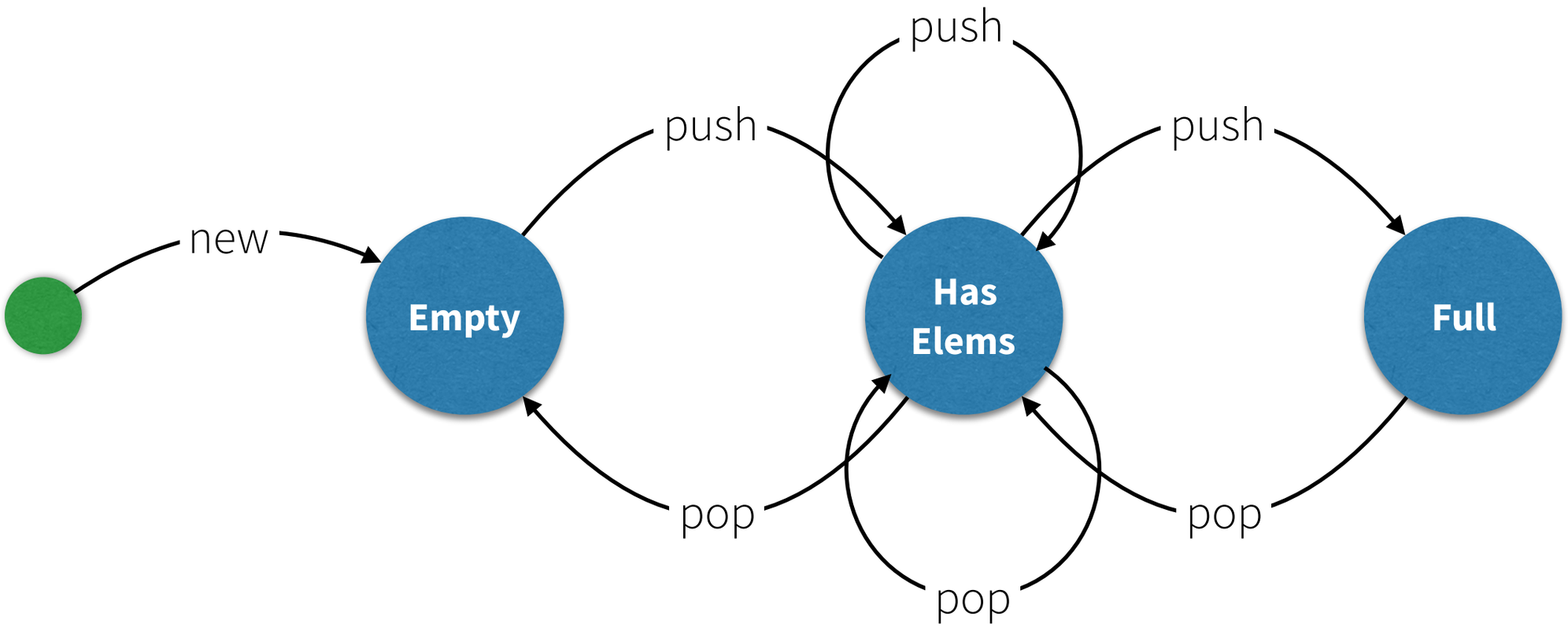
- **Naiv strategi:** plocka slumpmässiga indata
- **Problem:** är inte säkra på att vi prövar båda vägar genom `if`-satsen
- **Slutsats:** vi behöver nog med testfall för att pröva båda ”branches”

Funktionsorienterad testtäckning

```
int ioopm_stack_size(int_stack_t *s)
```

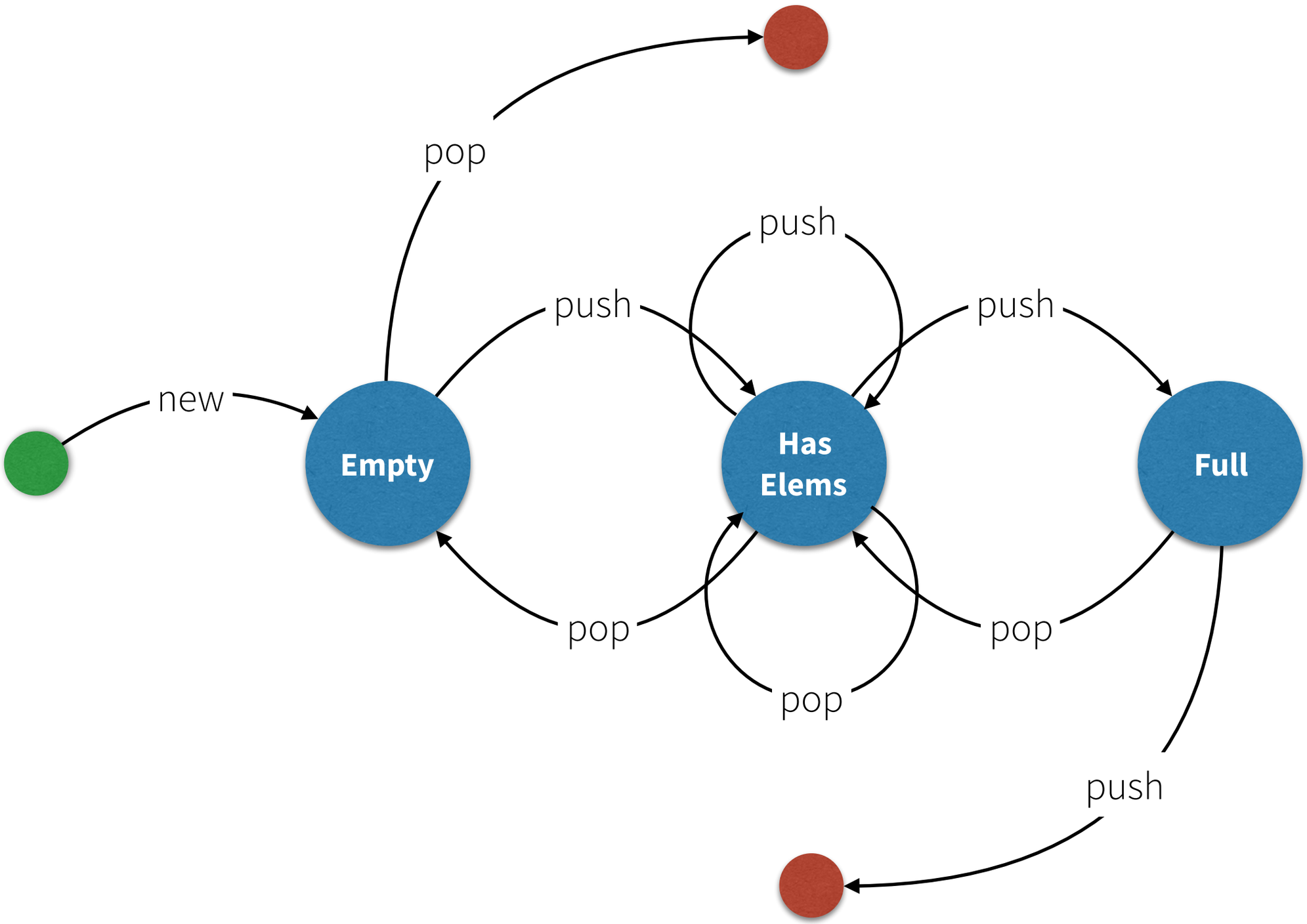
- **Naiv strategi:** skapa många stackar och pröva deras storlek
- **Problem:** inte säkert att vi prövar alla fall (tom stack, full stack, etc.)
- **Slutsats:** behöver fall som täcker alla möjliga *klasser av* input

Beteende-orienterad testtäckning

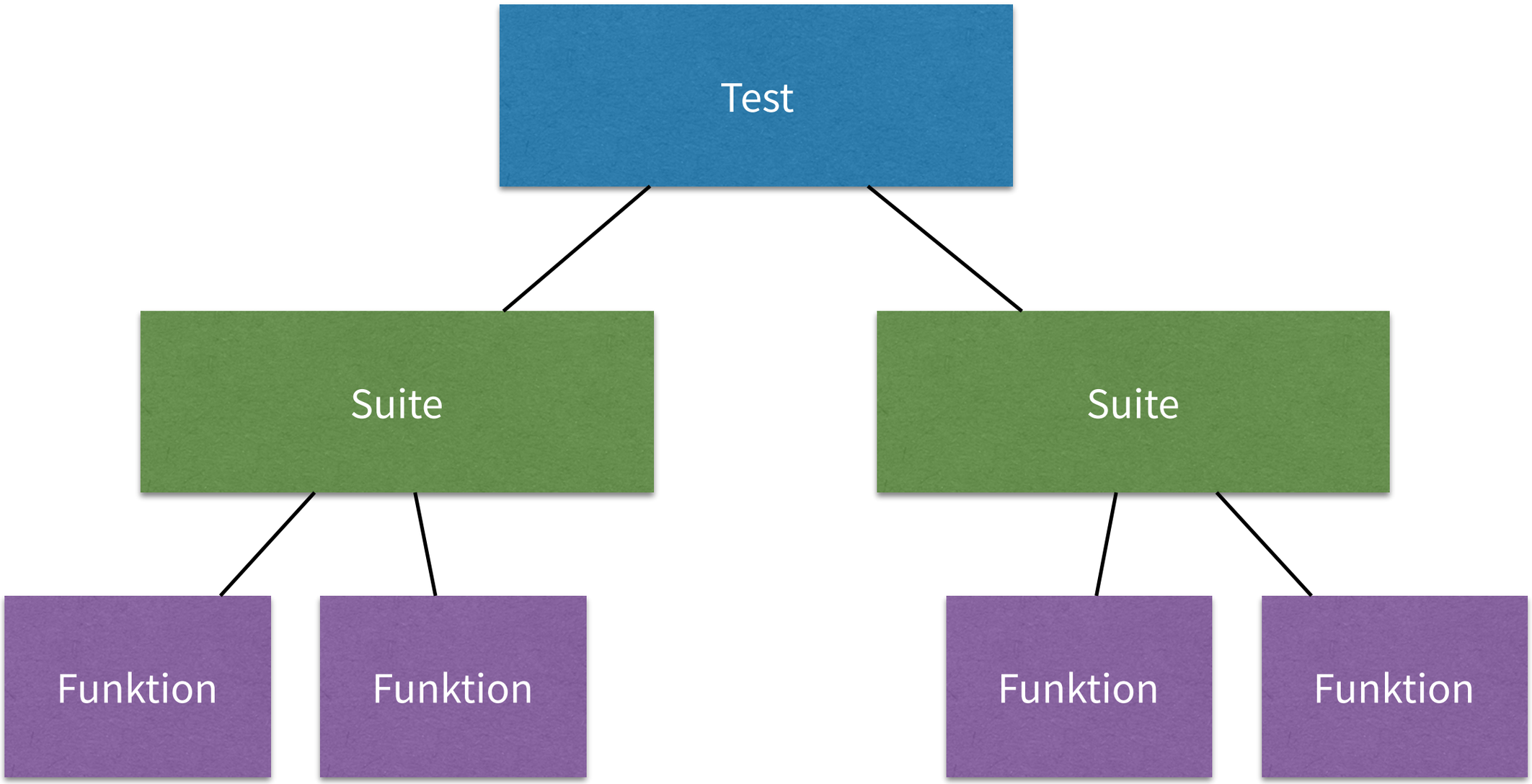


- **Naiv strategi:** plocka slumpmässiga värden, pusha, poppa och se att vi får tillbaka samma värden.
- **Problem:** är inte säkra på att vi prövar alla viktiga fall
- **Slutsats:** vi behöver testfall som prövar alla tillstånd och tillståndsövergångar

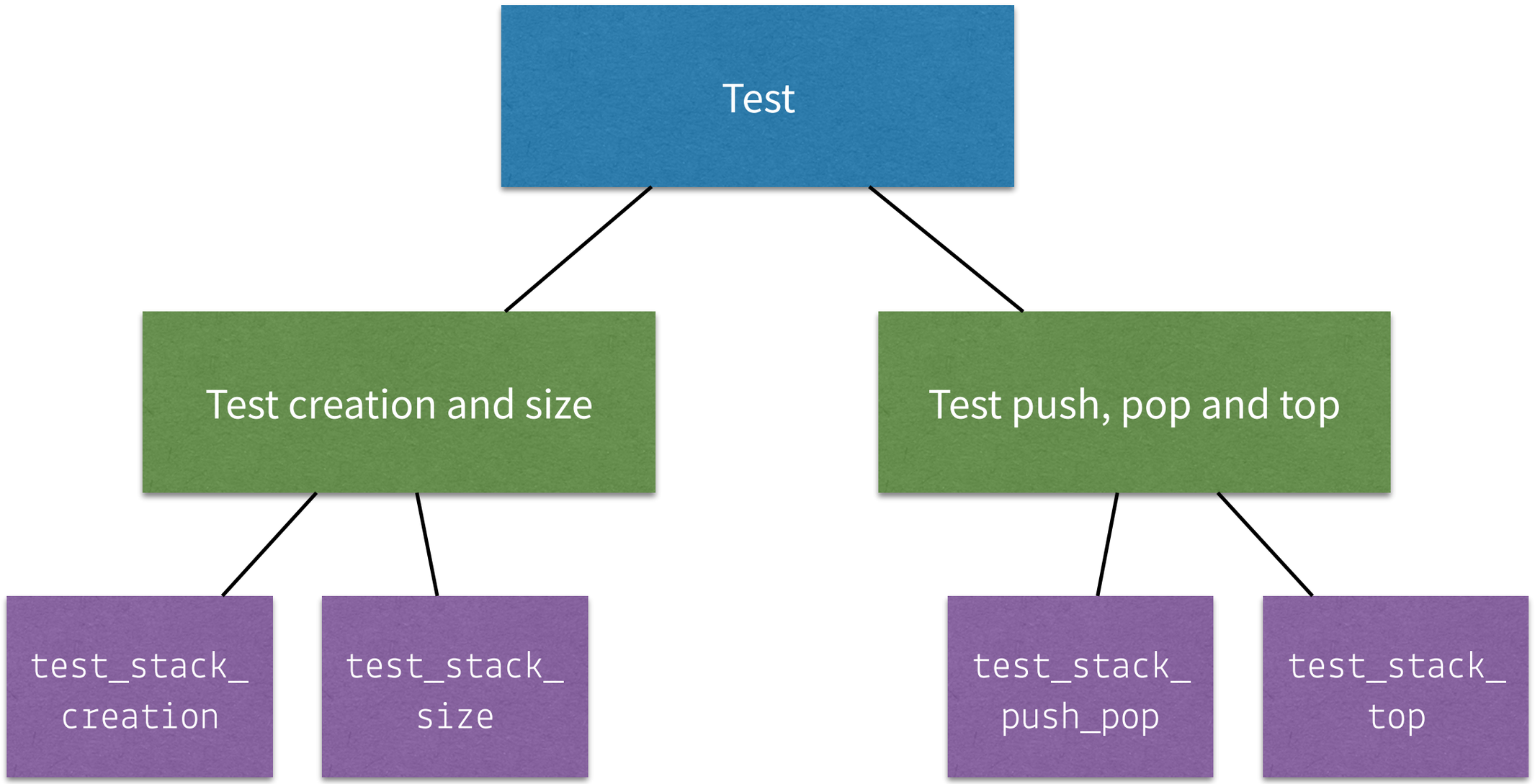
Beteende-orienterad testtäckning



Ett testprogram i CUnit



stack_test.c


Test
Test creation and size
Test push, pop and top
test_stack_creation
test_stack_size
test_stack_push_pop
test_stack_top

main()

```
int main(int argc, char *argv[])
{
  // Initialise
  CU_initialize_registry();

  // Set up suites and tests
  CU_pSuite creation = CU_add_suite("Test creation and height", NULL, NULL);
  CU_add_test(creation, "Creation", test_stack_creation);
  CU_add_test(creation, "Size", test_stack_size);

  CU_pSuite pushpoptop = CU_add_suite("Test push, pop and top", NULL, NULL);
  CU_add_test(pushpoptop, "Push and pop", test_stack_push_pop);
  CU_add_test(pushpoptop, "Top", test_stack_top);

  // Actually run tests
  CU_basic_run_tests();

  // Tear down
  CU_cleanup_registry();
  return 0;
}
```

Skapa en stack

```
void test_stack_creation()
{
  stack_t *s = ioopm_stack_create(10);

  CU_ASSERT_FALSE(s == NULL);
  CU_ASSERT_TRUE(ioopm_stack_size(s) == 0);

  ioopm_stack_destroy(s);
}
```

Stackhöjden

```
void test_stack_size()
{
  stack_t *s = ioopm_stack_create(10);

  CU_ASSERT_TRUE(ioopm_stack_size(s) == 0);
  ioopm_stack_push(s, 0);
  CU_ASSERT_TRUE(ioopm_stack_size(s) == 1);
  ioopm_stack_push(s, 0);
  CU_ASSERT_TRUE(ioopm_stack_size(s) == 2);
  ioopm_stack_pop(s);
  CU_ASSERT_TRUE(ioopm_stack_size(s) == 1);
  ioopm_stack_pop(s);
  CU_ASSERT_TRUE(ioopm_stack_size(s) == 0);

  ioopm_stack_destroy(s);
}
```

Lägga till och ta bort

```
void test_stack_push_pop()
{
  stack_t *s = ioopm_stack_create(10);
  const int values[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

  for (int i = sizeof(values) / sizeof(values[0]); i > 0; --i)
    {
      ioopm_stack_push(s, values[i - 1]);
    }

  for (int i = 0; i < sizeof(values) / sizeof(int); ++i)
    {
      CU_ASSERT_EQUAL(ioopm_stack_top(s), values[i]);
      CU_ASSERT_TRUE(ioopm_stack_pop(s));

    }

  ioopm_stack_destroy(s);
}
```

Titta på översta elementet

```
void test_stack_top(10)
{
  stack_t *s = ioopm_stack_create();

  ioopm_stack_push(s, 1);
  CU_ASSERT_TRUE(ioopm_stack_top(s) == 1);
  ioopm_stack_push(s, 20);
  CU_ASSERT_TRUE(ioopm_stack_top(s) == 20);
  ioopm_stack_pop(s);
  CU_ASSERT_TRUE(ioopm_stack_top(s) == 1);

  ioopm_stack_destroy(s);
}
```

```
$ gcc -o stack_test stack_test.c stack.c -lcunit


$ ./stack_test

     CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      2      2    n/a      0        0
               tests      4      4      4      0        0
             asserts     20     20     20      0      n/a

Elapsed time =    0.000 seconds
```

```
$ gcc -o stack_test stack_test.c stack.c -lcunit


$ ./stack_test

     CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      2      2    n/a      0        0
               tests      4      4      4      0        0
             asserts     20     20     20      0      n/a

Elapsed time =    0.000 seconds
```

```
$ gcc -o stack_test stack_test.c stack.c -lcunit


$ ./stack_test

     CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      2      2    n/a      0        0
               tests      4      4      4      0        0
             asserts     20     20     20      0      n/a

Elapsed time =    0.000 seconds
```

Sök eller skriv in namn på webbplats

CUnit - A Unit testing framework for C.
http://cunit.sourceforge.net/

Automated Test Run Results

Running Suite Test creation and height		
	Running test Creation ...	Passed
	Running test Height ...	Passed
Running Suite Test push, pop and top		
	Running test Push and pop ...	Passed
	Running test Top ...	Passed

Cumulative Summary for Run					
Type	**Total**	**Run**	**Succeeded**	**Failed**	**Inactive**
Suites	2	2	- NA -	0	0
Test Cases	4	4	4	0	0
Assertions	20	20	20	0	n/a

File Generated By CUnit v2.1-3 - Sun Sep 20 23:54:41 2015

Visa en meny

```
#include <CUnit/Automated.h>
CU_automated_run_tests(); // run and generate XML file
```

```
$ gcc --coverage -o stack_test stack_test.c stack.c -lcunit
```

```
$ gcc --coverage -o stack_test stack_test.c stack.c -lcunit

$ ./stack_test
...
```

```
$ gcc --coverage -o stack_test stack_test.c stack.c -lcunit

$ ./stack_test
...

$ gcov stack_test.c stack.c
File 'stack_test.c'
Lines executed:100.00% of 45
stack_test.c:creating 'stack_test.c.gcov'

File 'stack.c'
Lines executed:72.41% of 29
stack.c:creating 'stack.c.gcov'
```

```
$ gcov -abcfu stack.c
Function 'ioopm_stack_pop'
Lines executed:80.00% of 5
No branches
No calls

Function 'ioopm_stack_push'
Lines executed:80.00% of 5
No branches
No calls

Function 'ioopm_stack_size'
Lines executed:100.00% of 2
No branches
No calls

.....

File 'stack.c'
Lines executed:90.91% of 22
Branches executed:100.00% of 6
Taken at least once:50.00% of 6
No calls
Creating 'stack.c.gcov'
```

a = all blocks

b = branch probabilities

c = branch counts

f = function summaries

u = unconditional branches

```
$ gcov -abcfu stack.c
Function 'ioopm_stack_pop'
Lines executed:80.00% of 5
No branches
No calls

Function 'ioopm_stack_push'
Lines executed:80.00% of 5
No branches
No calls

Function 'ioopm_stack_size'
Lines executed:100.00% of 2
No branches
No calls

.....

File 'stack.c'
Lines executed:90.91% of 22
Branches executed:100.00% of 6
Taken at least once:50.00% of 6
No calls
Creating 'stack.c.gcov'
```

```c
bool ioopm_stack_pop(stack_t *s)
{
  if (s->height > 0)
    {
      s->height--;
      return true
    }
  else
    {
      return false;
    }
}
```

a = all blocks

b = branch probabilities

c = branch counts

f = function summaries

u = unconditional branches

```
$ more stack.c.gcov
        -:    0:Source:stack.c
        -:    0:Programs:2
        -:    1://#include <stdio.h>
        -:    2:#include <stdlib.h>
        -:    3:#include <stdbool.h>
        -:    4:#include "stack.h"
        -:    5:
........
       13:   44:bool ioopm_stack_pop(stack_t *s)
        -:   45:{
       13:   46:  if (s->height > 0)
        -:   47:    {
       13:   48:      s->height--;
       13:   49:      return true;
        -:   50:    }
        -:   51:  else
        -:   52:    {
    #####:   53:      return false;
        -:   54:    }
        -:   55:}
        -:   56:
        -:   57:
```

```
bool ioopm_stack_pop(stack_t *s)
{
  if (s->height > 0)
    {
      s->height--;
      return true
    }
  else
    {
      return false;
    }
}
```

```
$ lcov -c -d . -o stack.info
Capturing coverage data from .
Found gcov version: 7.4.0
Scanning . for .gcda files ...
Found 2 data files in .
Processing stack.gcda
Processing stack_test.gcda
Finished .info-file creation

$ genhtml stack.info -o stack-lcov
Reading data file stack.info
Found 2 entries.
Found common filename prefix "/home/lhe"
Writing .css and .png files.
Generating output.
Processing file stack/stack.c
Processing file stack/stack_test.c
Writing directory view page.
Overall coverage rate:
  lines......: 97.2% (70 of 72 lines)
  functions..: 100.0% (11 of 11 functions)
```

c = capture coverage data

d = directory to read from

o = output file

```
$ ls stack-lcov/
amber.png      glass.png           index-sort-l.html  stack/
emerald.png    index.html          ruby.png           updown.png
gcov.css       index-sort-f.html   snow.png
```

Öppna `index.html` i en webbläsare!

Klick

LCOV - stack.info

LCOV - code coverage report

		Hit	Total	Coverage
Current view:	top level			
Test:	stack.info	Lines:	70	72	97.2 %
Date:	2019-09-23 07:57:30	Functions:	11	11	100.0 %

Directory	Line Coverage			Functions	
stack		97.2 %	70 / 72	100.0 %	11 / 11

Generated by: LCOV version 1.14

Klick

LCOV - stack.info - stack

LCOV - code coverage report

		Hit	Total	Coverage
Current view:	top level - stack			
Test:	stack.info	Lines:	70	72	97.2 %
Date:	2019-09-23 07:57:30	Functions:	11	11	100.0 %

Filename	Line Coverage			Functions	
stack.c		90.9 %	20 / 22	100.0 %	6 / 6
stack_test.c		100.0 %	50 / 50	100.0 %	5 / 5

Generated by: LCOV version 1.14

LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level - stack - stack.c (source / functions)	**Lines:**	20	22	90.9 %
Test:	stack.info	**Functions:**	6	6	100.0 %
Date:	2019-09-23 07:57:30				

```
      Line data    Source code
         1              : //#include <stdio.h>
         2              : #include <stdlib.h>
         3              : #include <stdbool.h>
         4              : #include "stack.h"
         5              :


        44           26 : bool ioopm_stack_pop(stack_t *s)
        45              : {
        46           26 :   if (s->height > 0)
        47              :     {
        48           26 :       s->height--;
        49           26 :       return true;
        50              :     }
        51              :   else
        52              :     {
        53            0 :       return false;
        54              :     }
        55              : }
        56              :
        57              :
```

Öppna problem ännu så länge

- Hur kontrollerar vi sidoeffekter?

  T.ex. `ioopm_stack_destroy()`

  Utskrifter i terminalfönstret eller filer skrivna på hårddisken

Skapa era egna enhetstester

- Utgå gärna från `stack_test.c` som en mall för era egna tester
  (Titta här: `github.com/TobiasWrigstad/ioopm19`)
- Låt er inspireras av de `ASSERT`s som finns i CUnit-dokumentationen
- **Gränsvärden!**
- Tänk på att mäta code coverage för att få en uppfattning om testens kvalitet
- Automatisera allt! (Nästa föreläsning...)

Några tips vid test av länkad lista

- Tomma listan
- `f` i början, i mitten och i slutet

  `f` = borttagning, insättning, etc.

- Inkapsling

  Växer storleken som den skall?

  Testa sortering genom att göra insättning och kolla index

  Testa dubletthantering genom att kolla om storleken växer vid dublettinsättning

  …

Några tips vid test av binärt sökträd

- Tomma trädet
- `f` vid ingen förälder men barn, vid både förälder och barn, och bara förälder

  `f` = borttagning, insättning, etc.

- Inkapsling

  Växer djup och storlek som de skall?

  Testa balansering med hjälp av djupet

  Testa dubbletter med hjälp av storlek

  Kan tillhandahålla funktioner för att kontrollera ”layout” etc. utan direkt åtkomst

Exempel: testa lagerdatabas

- Börja med kraven!

  Dubletter?
  Kortaste avstånd?
  …etc.

- Den hypotetiska funktionen ”Packa en pall”

  Lägg i en vara, kolla att pris stämmer
  Lägg i en till, ändrades priset korrekt
  Vad händer vid borttaggning?
  …etc.

- Notera alla specialfall i din kod och testa extra för dem!

Vad är målet med testning?

- Att hitta så många fel som möjligt i programmet?
- Att visa att programmet är felfritt?
- Något annat?

Testdriven utveckling

- Vid implementation av funktionen f, ta reda på vad f skall göra

  Skriv ned detta i termer av tester

  Gör detta innan du börjar implementera funktionen

- Börja implementera, målet är att få testerna att passera

  Varje testfall kan vara en enhet att implementera

  Du har tydliga kriterier för när du är klar

- **Svårt att göra i praktiken — öva och ge inte upp!**

Föreläsning 8

Lars-Henrik Eriksson

Föreläsningsbilderna är baserade på bilder gjorda av Tobias Wrigstad.

Automatisering och verktyg profilering och optimering

stack.h

```
typedef struct stack stack_t;

struct stack
{
  int height;
  int maxsize;
  int values[0];
};

......
```

stack.c

```
#include <stdlib.h>
#include <stdbool.h>
#include "stack.h"

stack_t *ioopm_stack_create(int maxsize)
{
  stack_t *result =
            malloc(sizeof(stack_t) +
                   maxsize*sizeof(int));
  if (result)
    {
      *result = (stack_t){ .height = 0,
                           .maxsize = maxsize };
    }
  return result;
}

void ioopm_stack_destroy(stack_t *s)
{
  free(s);
}

int ioopm_stack_top(stack_t *s)
{
  return s->values[s->height-1];
}

int ioopm_stack_size(stack_t *s)
{
  return s->height;
}
```

```
bool ioopm_stack_push(stack_t *s, int v)
{
  if (s->height < s->maxsize)
    {
      s->values[s->height++] = v;
      return true;
    }
  else
    {
      return false;
    }
}

bool ioopm_stack_pop(stack_t *s)
{
  if (s->height > 0)
    {
      s->height--;
      return true;
    }
  else
    {
      return false;
    }
}
```

```c
int main(int argc, char *argv[])
{
  // Initialise
  CU_initialize_registry();

  // Set up suites and tests
  CU_pSuite creation = CU_add_suite("Test creation and height", NULL, NULL);
  CU_add_test(creation, "Creation", test_stack_creation);
  CU_add_test(creation, "Size", test_stack_size);

  CU_pSuite pushpoptop = CU_add_suite("Test push, pop and top", NULL, NULL);
  CU_add_test(pushpoptop, "Push and pop", test_stack_push_pop);
  CU_add_test(pushpoptop, "Top", test_stack_top);

  // Actually run tests
  CU_basic_run_tests();

  // Tear down
  CU_cleanup_registry();
  return 0;
}
```

```c
void test_stack_push_pop()
{
  stack_t *s = ioopm_stack_create(10);
  const int values[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

  for (int i = sizeof(values) / sizeof(values[0]); i > 0; --i)
    {
      ioopm_stack_push(s, values[i - 1]);
    }

  for (int i = 0; i < sizeof(values) / sizeof(int); ++i)
    {
      CU_ASSERT_EQUAL(ioopm_stack_top(s), values[i]);
      CU_ASSERT_TRUE(ioopm_stack_pop(s));

    }
     ioopm_stack_destroy(s);
}
```

stack.h

```
typedef struct stack stack_t;

struct stack
{
  int height;
  int maxsize;
  int values[0];
};

......
```

stack.c

```
#include <stdlib.h>
#include <stdbool.h>
#include "stack.h"

stack_t *ioopm_stack_create(int maxsize)
{
  stack_t *result =
            malloc(sizeof(stack_t) +
                   maxsize*sizeof(int));
  if (result)
    {
      *result = (stack_t){ .height = 0,
                           .maxsize = maxsize };
    }
  return result;
}

void ioopm_stack_destroy(stack_t *s)
{
  free(s);
}

int ioopm_stack_top(stack_t *s)
{
  return s->values[s->height-1];
}

int ioopm_stack_size(stack_t *s)
{
  return s->height;
}
```

```
bool ioopm_stack_push(stack_t *s, int v)
{
  printf("Entering push, height = %d\n",
         s->height);
  if (s->height < s->maxsize)
    {
      s->values[s->height++] = v;
      printf("Returned true\n");
      return true;
    }
  else
    {
      printf("Returned false\n");
      return false;
    }
}

bool ioopm_stack_pop(stack_t *s)
{
  if (s->height > 0)
    {
      s->height--;
      return true;
    }
  else
    {
      return false;
    }
}
```

Felsökning med gdb

```
$ gcc -g -o stack_test stack_test.c stack.c -lcunit
$ gdb stack_test
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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"...
Reading symbols from stack_test...done.
(gdb)
```

```
(gdb) break test_stack_push_pop
Breakpoint 1 at 0xbdd: file stack_test.c, line 32.
(gdb) run
Starting program: /home/lhe/stack/a.out
    CUnit - A unit testing framework for C - Versi
     http://cunit.sourceforge.net/
Breakpoint 1, test_stack_push_pop () at stack_test.c:32
32  {
(gdb) next
33    stack_t *s = ioopm_stack_create(10);
(gdb) next
34    const int values[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
(gdb) print s
$1 = (stack_t *) 0x555555757510
(gdb) explore s
's' is a pointer to a value of type 'stack_t'
Continue exploring it as a pointer to a single value [y/n]: y
The value of '*s' is of type 'stack_t' which is a typedef of type 'struct stack'
The value of '*s' is a struct/class of type 'struct stack' with the following
fields:
   height = 0 .. (Value of type 'int')
  maxsize = 10 .. (Value of type 'int')
   values = <Enter 2 to explore this field of type 'int []'>
Enter the field number of choice:
(gdb) print s->maxsize
$2 = 10
```

```c
void test_stack_push_pop()
{
  stack_t *s = ioopm_stack_create(10);
  const int values[] =
     { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

```c
typedef struct stack stack_t;
struct stack
{
  int height;
  int maxsize;
  int values[0];
};
```

Kör till nästa rad i *samma* fil

```
for (int i = sizeof(values) / sizeof(values[0]); i > 0; --i)
  {
    ioopm_stack_push(s, values[i - 1]);
  }
```

```
(gdb) next
36    for (int i = sizeof(values) / sizeof(values[0]); i > 0; --i)
(gdb) next
38        ioopm_stack_push(s, values[i - 1]);
(gdb) step
ioopm_stack_push (s=0x555555757510, v=9) at stack.c:33
33    if (s->height < s->maxsize)
(gdb) print v
$3 = 9
(gdb) set var v = 8
(gdb) finish
Run till exit from #0  ioopm_stack_push (s=0x555555757510, v=8) at stack.c:33
test_stack_push_pop () at stack_test.c:36
36    for (int i = sizeof(values) / sizeof(values[0]); i > 0; --i)
Value returned is $4 = true
```

Kör till nästa rad i *någon* fil

Kör färdigt aktuell funktion

```
bool ioopm_stack_push(stack_t *s, int v)
{
  if (s->height < s->maxsize)
    {
      s->values[s->height++] = v;
      return true;
```

```
  for (int i = 0; i < sizeof(values) / sizeof(int); ++i)
    {
      CU_ASSERT_TRUE(ioopm_stack_top(s) == values[i]);
      ioopm_stack_pop(s);
    }
```

```
(gdb) break ioopm_stack_pop
Breakpoint 2 at 0x555555554f9a: file stack.c, line 46.
(gdb) continue
Continuing.
Breakpoint 2, ioopm_stack_pop (s=0x555555757510) at stack.c:46
46    if (s->height > 0)
(gdb) print s->height
$6 = 10
(gdb) c
Continuing.
Breakpoint 2, ioopm_stack_pop (s=0x555555757510) at s
46    if (s->height > 0)
(gdb) print s->height
$7 = 9
(gdb) c
Continuing.
Breakpoint 2, ioopm_stack_pop (s=0x555555757510) at stack.c:46
46    if (s->height > 0)
(gdb) print s->height
$8 = 8
```

```
bool ioopm_stack_pop(stack_t *s)
{
  if (s->height > 0)
    {
      s->height--;
      return true;
    }
  else
    {
      return false;
    }
}
```

```
(gdb) clear ioopm_stack_pop
Deleted breakpoint 2
(gdb) break ioopm_stack_pop if s->height == 3
Breakpoint 3 at 0x555555554f9a: file stack.c, line 46.
(gdb) c
Continuing.
Breakpoint 3, ioopm_stack_pop (s=0x555555757510) at st
46    if (s->height > 0)
(gdb) print s->height
$9 = 3
(gdb) print s->values[$9]
$10 = 6
(gdb) c
Continuing.
```

```c
bool ioopm_stack_pop(stack_t *s)
{
  if (s->height > 0)
    {
      s->height--;
      return true;
    }
  else
    {
      return false;
    }
}
```

```
Suite Test push, pop and top, Test Push and pop had failures:
    1. stack_test.c:43  - CU_ASSERT_TRUE(ioopm_stack_top(s) == values[i])

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      2      2    n/a      0        0
               tests      4      4      3      1        0
             asserts     20     20     19      1      n/a

Elapsed time =    0.003 seconds
[Inferior 1 (process 8739) exited normally]
(gdb) quit
```

```
$ emacs stack.c
$ gcc -g -o stack_test stack_test.c stack.c -lcunit
$ gdb ./stack_test…
$ emacs stack.c
$ gcc -g -o stack_test stack_test.c stack.c -lcunit
$ gdb ./stack_test…
$ emacs stack.c
$ gcc -g -o stack_test stack_test.c stack.c -lcunit
$ gdb ./stack_test…
$ emacs stack.c
$ gcc -g -o stack_test stack_test.c stack.c -lcunit
$ gdb ./stack_test…
$ emacs stack.c
$ gcc -g -o stack_test stack_test.c stack.c -lcunit
$ gdb ./stack_test…
$ emacs stack.c
$ gcc -g -o stack_test stack_test.c stack.c -lcunit
$ gdb ./stack_test…
$ emacs stack.c
$ gcc -g -o stack_test stack_test.c stack.c -lcunit
$ gdb ./stack_test…
$ emacs stack.c
$ gcc -g -o stack_test stack_test.c stack.c -lcunit
$ gdb ./stack_test…
$ emacs stack.c
$ gcc -g -o stack_test stack_test.c stack.c -lcunit
```

Kommandofiler i shell (scripts)

Vilket program (shell) skall tolka filen?

```
#!/bin/bash
emacs stack.c
gcc -g -o stack_test stack.c stack_test.c -lcunit
gdb stack_test
```

work

- Körs som vilket program som helst, t.ex.
  `$ ./work`
- Glöm inte att göra `chmod u+x work` (eller `chmod ugo+x` work)

Man kan göra mycket med shellscripts

Första argument till scriptet

```
#!/bin/bash
scp -p "$1" jeltz.it.uu.se:
```

tosolaris

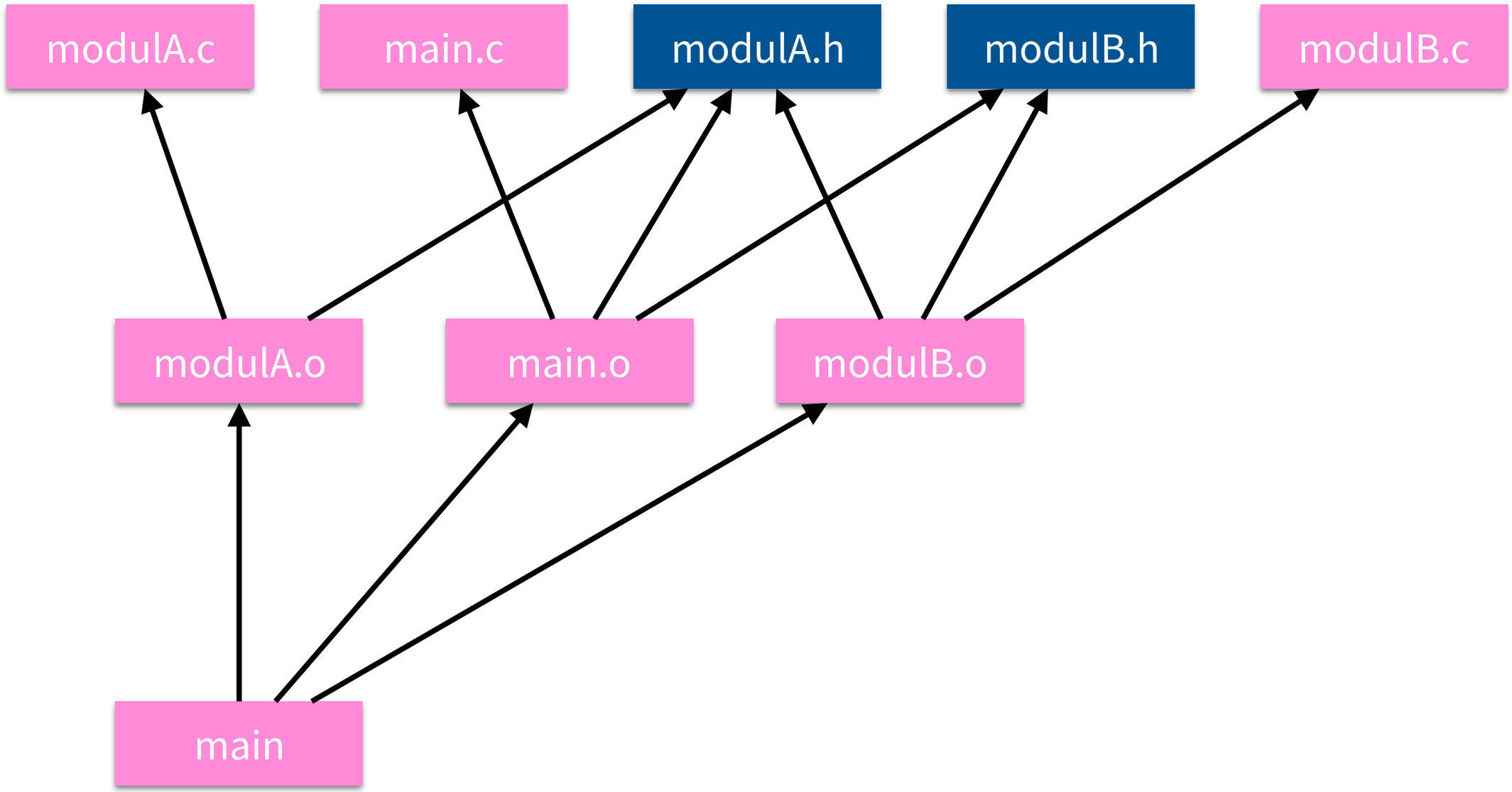
```
$ ./tosolaris stack.c
```

Alla argument till scriptet

```
#!/bin/bash
# Make backup copies of all argument files
for file in $@; do
    echo "$file"
    cp -p "$file" "$file.backup"
done
```

backup

```
$ ls
backup stack.c  stack.h  stack_test.c
$ ./backup *.c
stack.c
stack_test.c
$ ls
backup    stack.c      stack.c.backup
stack.h   stack_test.c stack_test.c.backup
```



Hur håller man reda på detta?

Byggverktyg

- Ett program består ofta av många filer
- Vi vill kunna bygga programmet på olika sätt

  e.x: för produktion (-O2), för test (-c, -g), etc.

- Vi vill bara bygga om de filer som behöver byggas om efter en ändring

  för att minimera byggtiden

- Vi vill enkelt kunna ändra på vilka flaggor som används för att bygga programmet

  e.x.: lägga till en länkflagga

Makefiler

mål

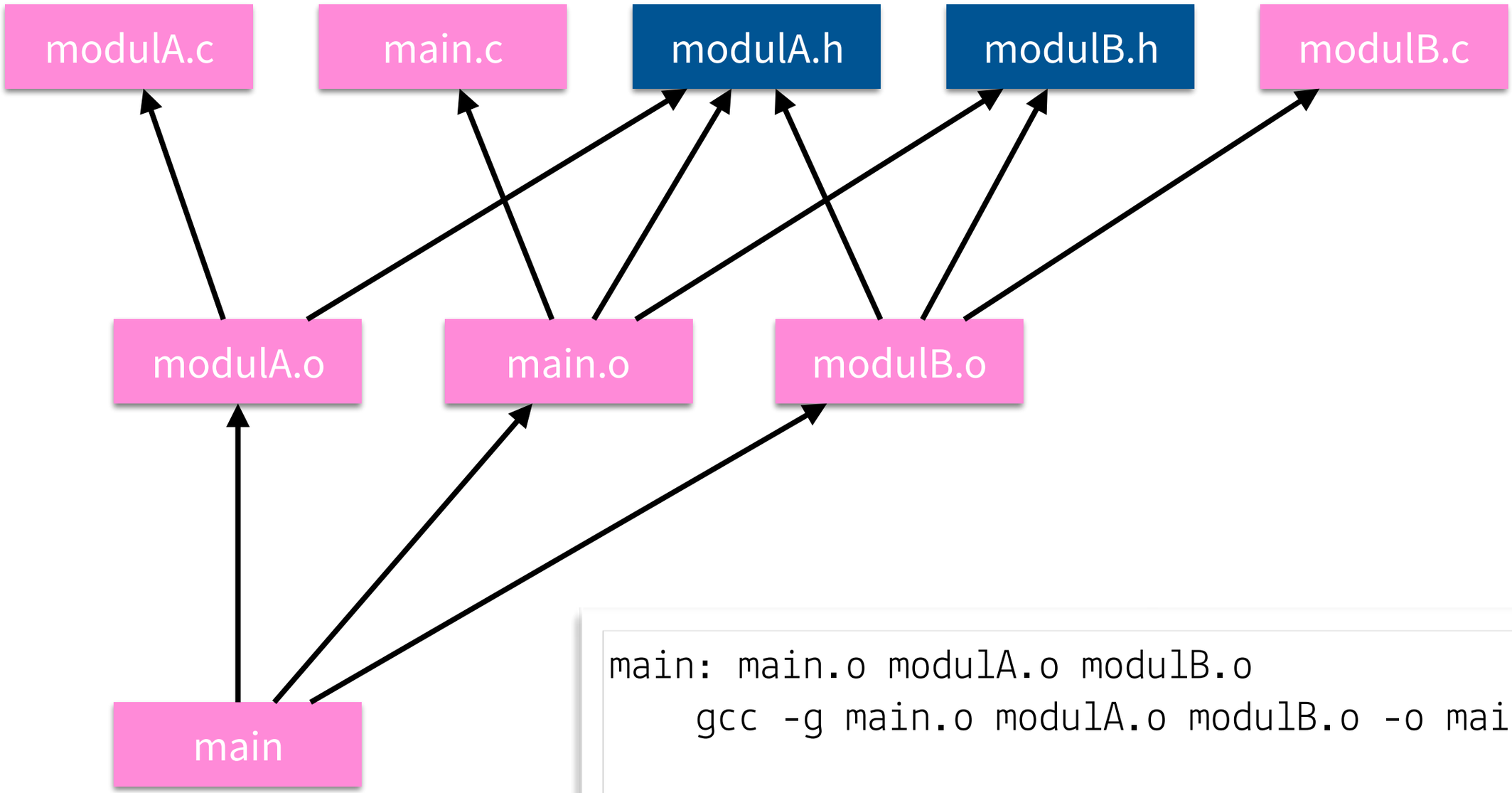
beroenden (ofta målfiler i andra mål)

```
main:   main.c sub.c sub.h
	gcc -g main.c sub.c -o main
```

OBS! TAB-tecken här.
Inte blanktecken!

kommando som utförs om målfilen inte finns eller är äldre än någon av de filer den är beroende av

Döps till `Makefile`, kör med `make`

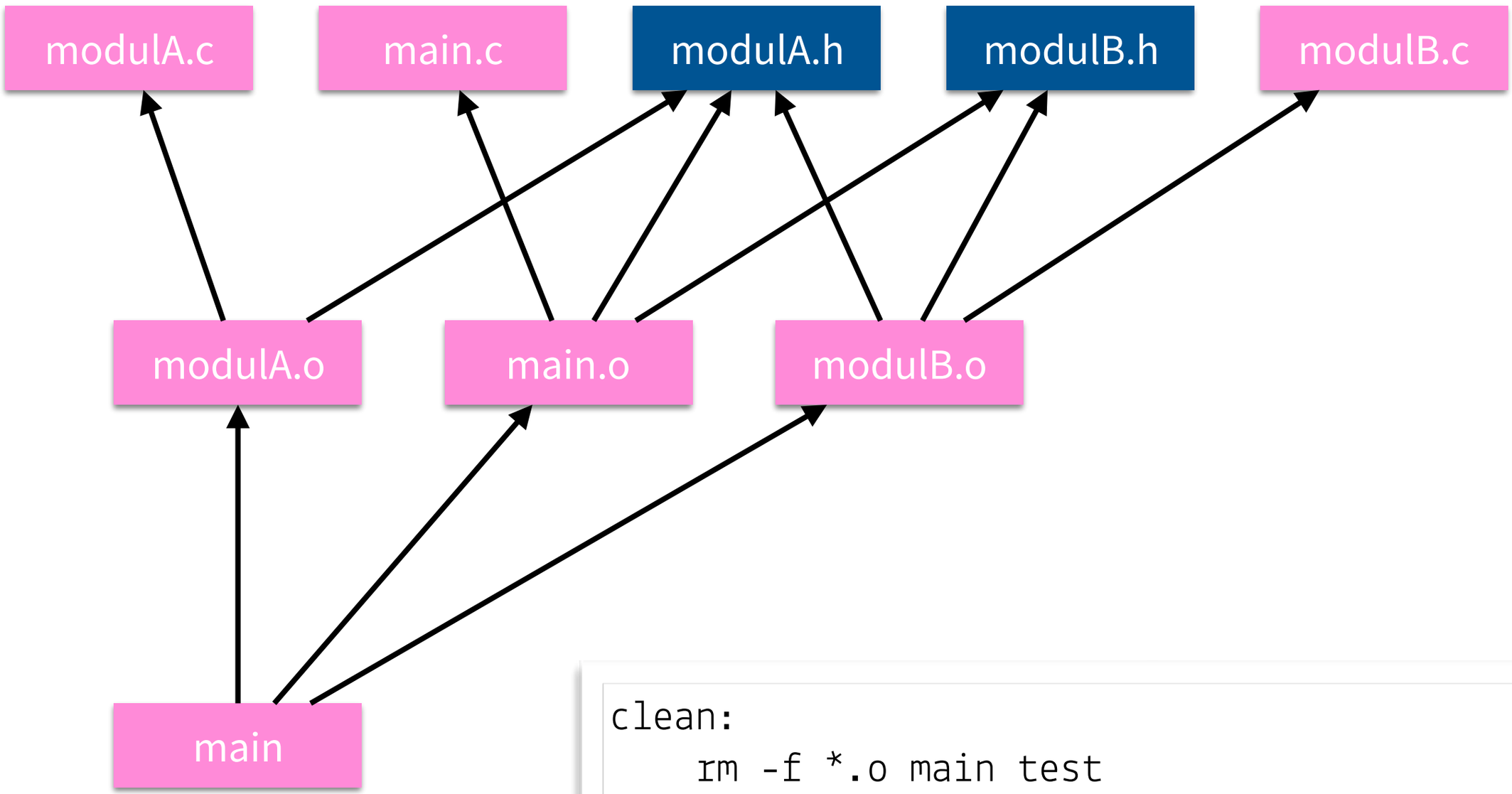


Exempel:

```
make main.o
make main
make
```

```
main: main.o modulA.o modulB.o
    gcc -g main.o modulA.o modulB.o -o main

main.o: main.c modulA.h modulB.h
    gcc -c -g -Wall main.c

modulA.o: modulA.c modulA.h
    gcc -c -g -Wall modulA.c

modulB.o: modulB.c modulA.h modulB.h
    gcc -c -g -Wall modulB.c
```

Makefile



Exempel:

```
make test
make clean
```

```
clean:
    rm -f *.o main test

memtest: test
    valgrind --leak-check=full ./test

test: modulA.o modulB.o tests.c
    gcc modulA.o modulB.o tests.c -o test
    ./test
```

Makefile

Variabler och wildcards

Det matchande namnet

Matchar filnamn

Alla beroenden som var nyare än målet

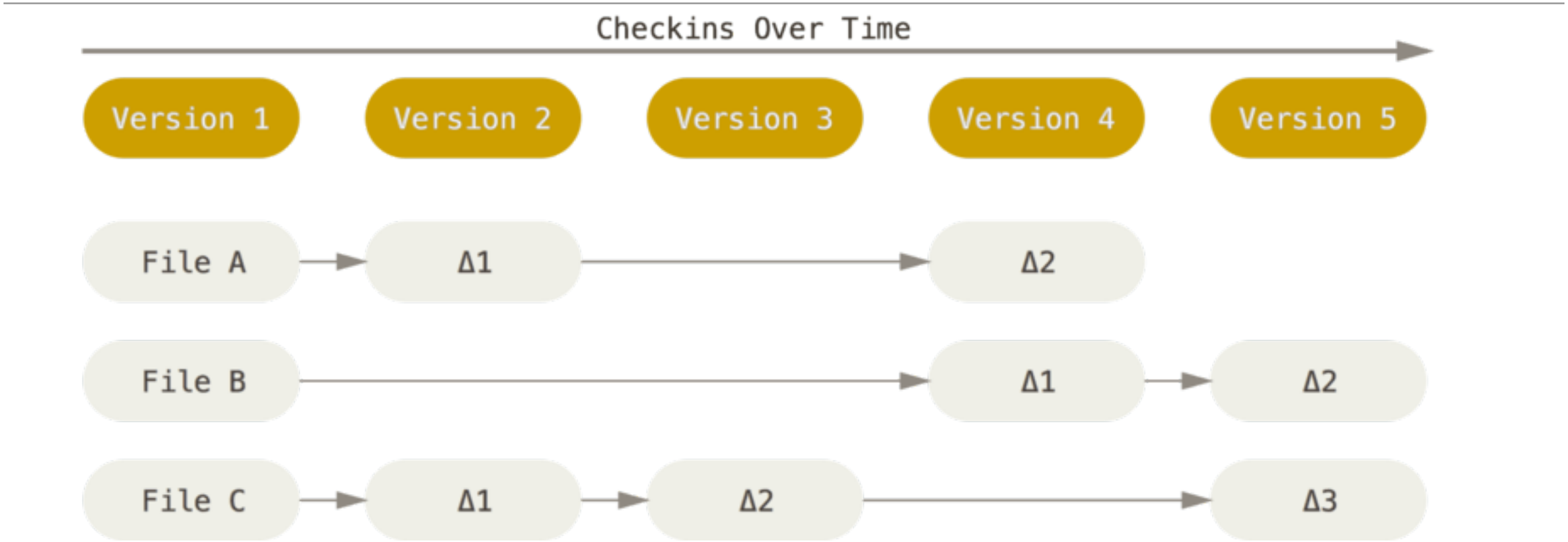
Namn på målet

```
main: myprog

C_COMPILER      = gcc
C_OPTIONS       = -Wall -pedantic -g
C_LINK_OPTIONS  = -lm
CUNIT_LINK      = -lcunit

%.o:  %.c
  $(C_COMPILER) $(C_OPTIONS) $? -c

myprog: file1.o file2.o file3.o
  $(C_COMPILER) $(C_LINK_OPTIONS) $? -o $@

test1: mytests1.o file1.o
  $(C_COMPILER) $(C_LINK_OPTIONS) $(CUNIT_LINK) $? -o $@

clean:
  rm -f *.o myprog
```

Make är inte världens bästa byggverktyg

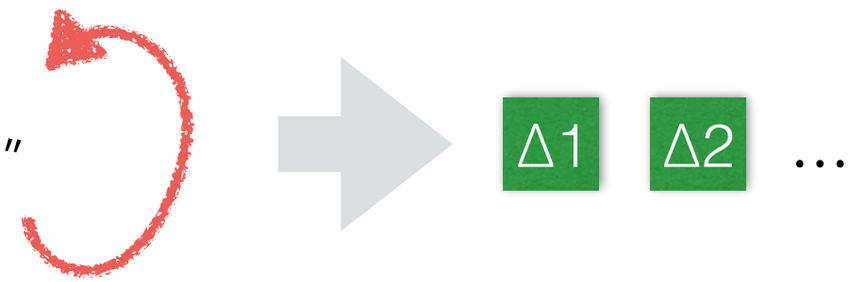
- Funkar jättebra för t.ex. C
- Funkar inte så bra för Java, t.ex. — för att namnstandarden inte funkar
- Finns många alternativ: cmake, premake, ant, etc.
- Korsa den bron när du kommer till den — nu räcker make fint!

Versionshantering med git

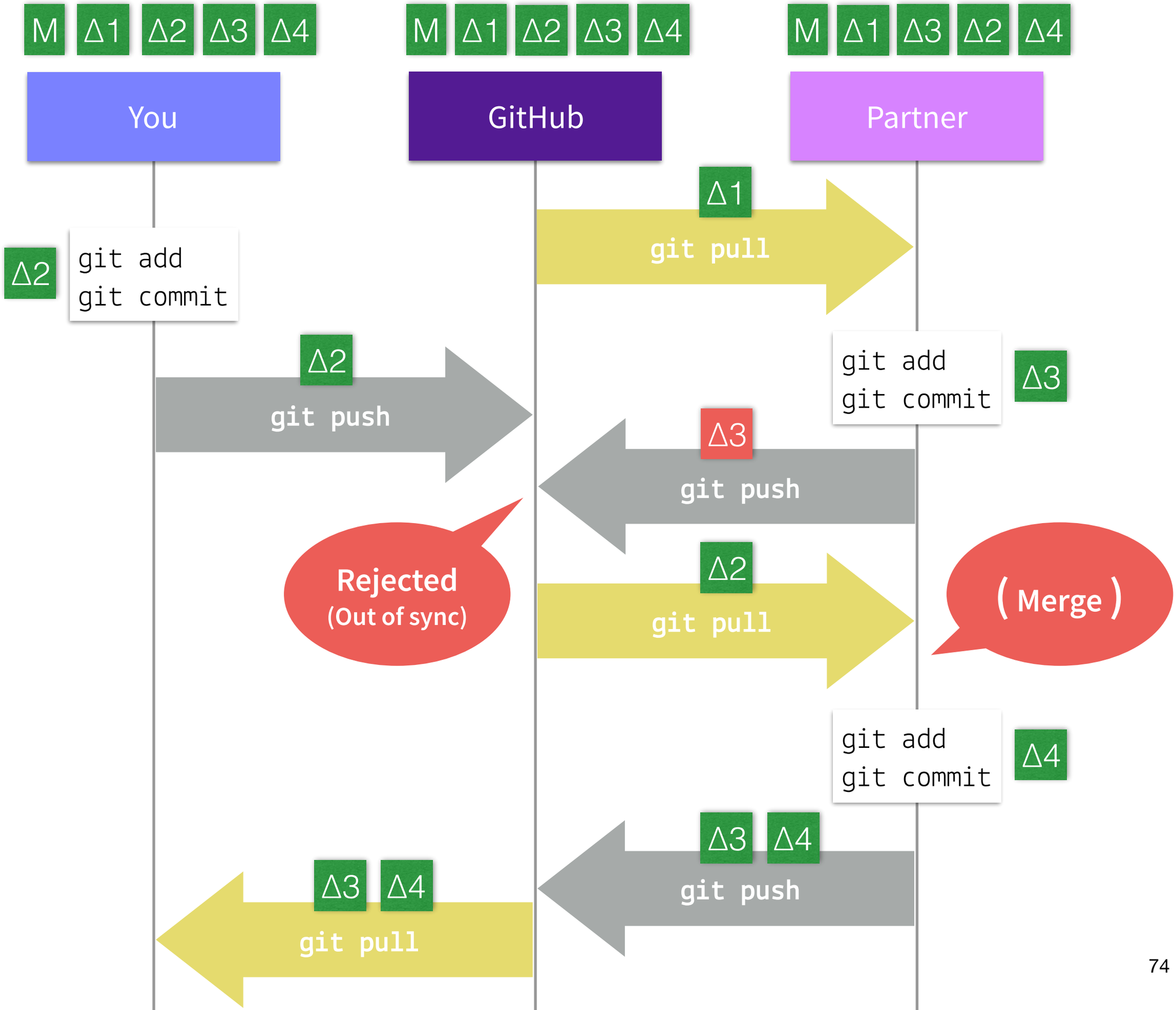


Arbetsflöde:

```
$ git add file.c file.h
$ git commit -m "Explanatory message"
```



https://git-scm.com/book/en/v2/Getting-Started-Git-Basics


M Δ1 Δ2 Δ3 Δ4
M Δ1 Δ2 Δ3 Δ4
M Δ1 Δ3 Δ2 Δ4
You
GitHub
Partner
Δ1
git pull
Δ2
git add
git commit
Δ2
git push
git add
git commit
Δ3
Δ3
git push
Rejected
(Out of sync)
Δ2
git pull
(Merge)
git add
git commit
Δ4
Δ3 Δ4
git push
Δ3 Δ4
git pull

Tre regler för optimering

- Den första regeln för optimering är

  Gör det inte

- Den andra regeln för optimering är

  Gör det inte förrän du har klara bevis på att optimering är nödvändigt

- Den tredje regeln för optimering är

  Du kan inte räkna ut vad som bör optimeras — du behöver profileringdata

Följdsatser

- Följdsats §1

  Optimering tenderar att göra bra kod oläslig

- Följdsats §2

  Kostnaden för programmets prestandaökning betalas igen i underhåll

Profilering

- CPU-tid vs. wall-clock time

  Jitter — vad händer i datorn i övrigt
  Klockans upplösning

- Instrumentering

  Heisenbergeffekter (profilering påverkar i sig prestanda)

- Sampling

  Spara data om exekvering löpande, t.ex. ”var exekverar programmet just nu”

- Tracing

  Programmet spelar in nyckelhändelser i en logg

Att mäta tid i en dator

real (wall clock) time

= user time *(time executing instructions in the user process)*

= system time *(time executing instructions in kernel on behalf of user process)*

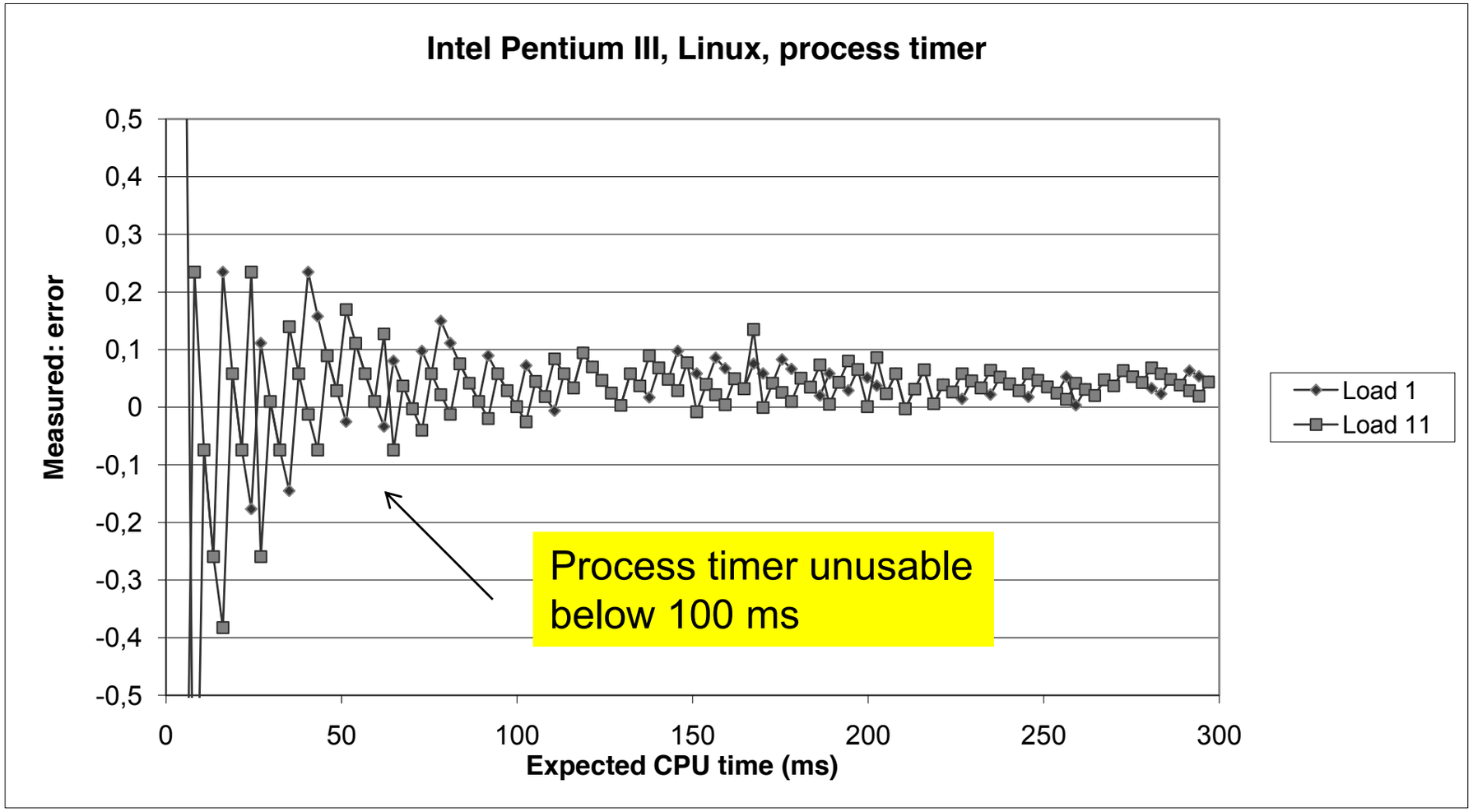
= some other user's time *(time executing instructions in different user's process)*

+ **+** **= real (wall clock) time**

***We will use the word "time" to refer to user time*.**

cumulative user time

Hur mycket *är* klockan?



Profilering med `gprof`

- Använda `gprof` för att ta fram en profil för ett program

```
$ gcc -pg myprog.c -o myprog

$ ./myprog

$ gprof gmon.out myprog > profile.txt
```

Skapas vid körningen av `myprog`

```
$ more profile.txt
```

Total tid i funktionen

Antal anrop av funktionen

Genomsnittlig tid i funktionen per anrop

...inklusive anropade funktioner

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 16.01      1.30     1.30 32001519     0.00     0.00  treeset_insert
 15.83      2.59     1.29 15999889     0.00     0.00  worker_commit
 12.19      3.58     0.99  1599989     0.00     0.00  treeset_reset
 10.10      4.40     0.82                             array_get
  9.91      5.20     0.81 15999890     0.00     0.00  worker_generate_connection
  8.99      5.93     0.73                             run
  6.65      6.47     0.54  1599989     0.00     0.00  worker_add_full_node
  6.04      6.96     0.49 16001630     0.00     0.00  random_rand
  4.31      7.31     0.35 16001630     0.00     0.00  random_coin_flip
  2.40      7.51     0.20 15999889     0.00     0.00  dependencies_fire
  2.34      7.70     0.19  8000865     0.00     0.00  worker_resolved_backward_dependency
  2.09      7.87     0.17  8000765     0.00     0.00  worker_resolved_node
  0.86      7.94     0.07                             run_thread
  0.62      7.99     0.05        1     0.05     6.37  worker_explore
  0.49      8.03     0.04                             cpu_core_pause
  0.37      8.06     0.03                             array_set
  0.12      8.07     0.01        1     0.01     0.01  random__init
  0.12      8.08     0.01        1     0.01     0.01  treeset__init
  0.12      8.09     0.01                             array_size
  0.12      8.10     0.01                             messageq_push
  0.12      8.11     0.01                             pony_thread_create
  0.12      8.12     0.01                             quiescent
  0.06      8.12     0.01                             dependencies_insert
```

Profilering och optimering under kursen

- Profilera inte ett (komplett) interaktivt program! (t.ex. lagerhanteraren)

  99,999% av dess tid går åt till att vänta på att användaren trycker på en tangent

- Välj någon delfunktion

  T.ex. funktionerna som lägger till och tar bort varor.

  Skriv ett program som lägger in 1 M varor och tar bort 100 k varor

  Profilera detta program!

Optimering

- De viktigaste optimeringarna är i regel algoritmiska!

  Använd ett bättre sätt att räkna fram det värdet du vill ha

```
uint64_t fib(const int n)
{
  switch (n)
    {
    case 0:  return 0;
    case 1:  return 1;
    default: return fib(n - 1) + fib(n - 2);
    }
}
```

```
uint64_t fib(const int n)
{
  uint64_t acc1, acc2, temp;

  acc1 = 0;
  acc2 = 1;

  for (int i = 0; i < n; ++i)
    {
      temp = acc2;
      acc2 += acc1;
      acc1 = temp;
    }

  return acc1;
}
```

Låt kompilatorn hjälpa till!

```
a[x+1] = b[x+1]
```

~~
```
int x1 = x+1;
a[x1] = b[x1]
```
~~

```
for (int i=0; i<10; i++)
  {
    a[i+k*4] = 0;
  }
```

~~
```
int k4 = k*4;
for (int i=0; i<10; i++)
  {
    a[i+k4] = 0;
  }
```
~~

```
gcc -O prog.c

gcc -O2 prog.c

gcc -O3 prog.c
```

Optimera för minne eller hastighet?

- I regel svårt att göra båda!