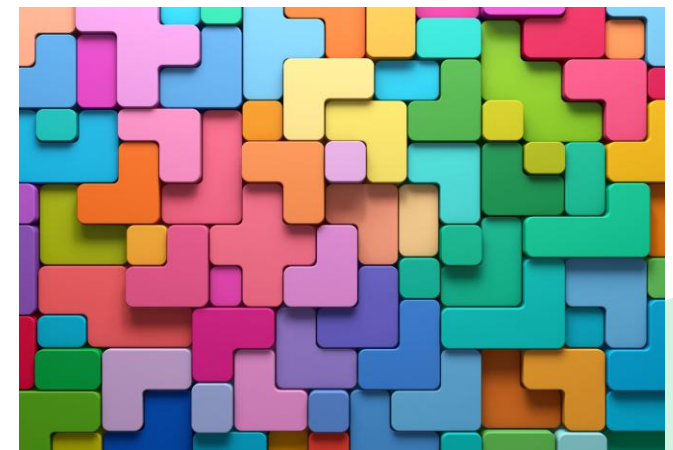
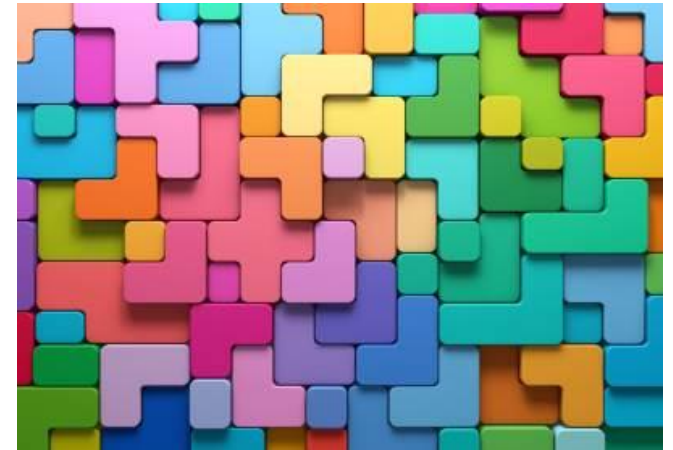


PROFILERING OG OPTIMALISERING

FORELESNING 22

FREDAG 7/11



(bilder generert av bing image creator)



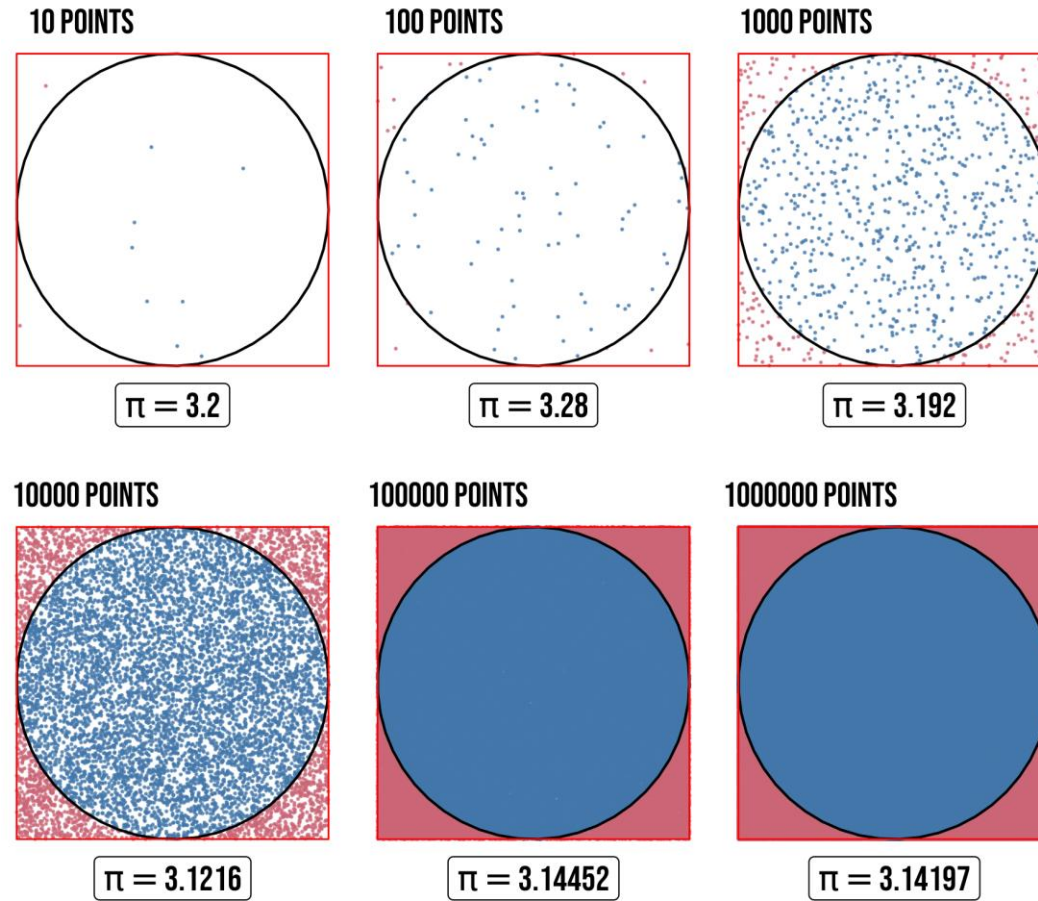
**L I V E K O D I N G :
S T I G E S P I L L**

○ Læremål: Algoritmeanalyse og optimalisering

- Å kunne analysere hvor rask en algoritme er (viktig når vi får enorme mengder data)
- Profilerings: finne flaskehalser (10% av koden bruker vanligvis 90% av tiden)
- (+ mer i ukene som kommer)



○ Monte Carlo-integrasjon



LIVEKODING: METODER FOR Å FINNE PI



○ Når skal vi optimalisere kode?

1. Få det til å virke

- Ikke tenk på kjøretid her
- Test grundig og nøye at det virker

2. Få det til å virke *bra*

- Gjør koden leselig og enkel å vedlikeholde (elegant kode)
- Del opp i klasser/funksjoner, lag bedre variabelnavn osv.

3. Få det til å virke bra og *raskt*

- Finn flaskehalsene og se om de kan gjøres raskere (må testes)
- Algoritmeanalyse og evt. blandet programmering



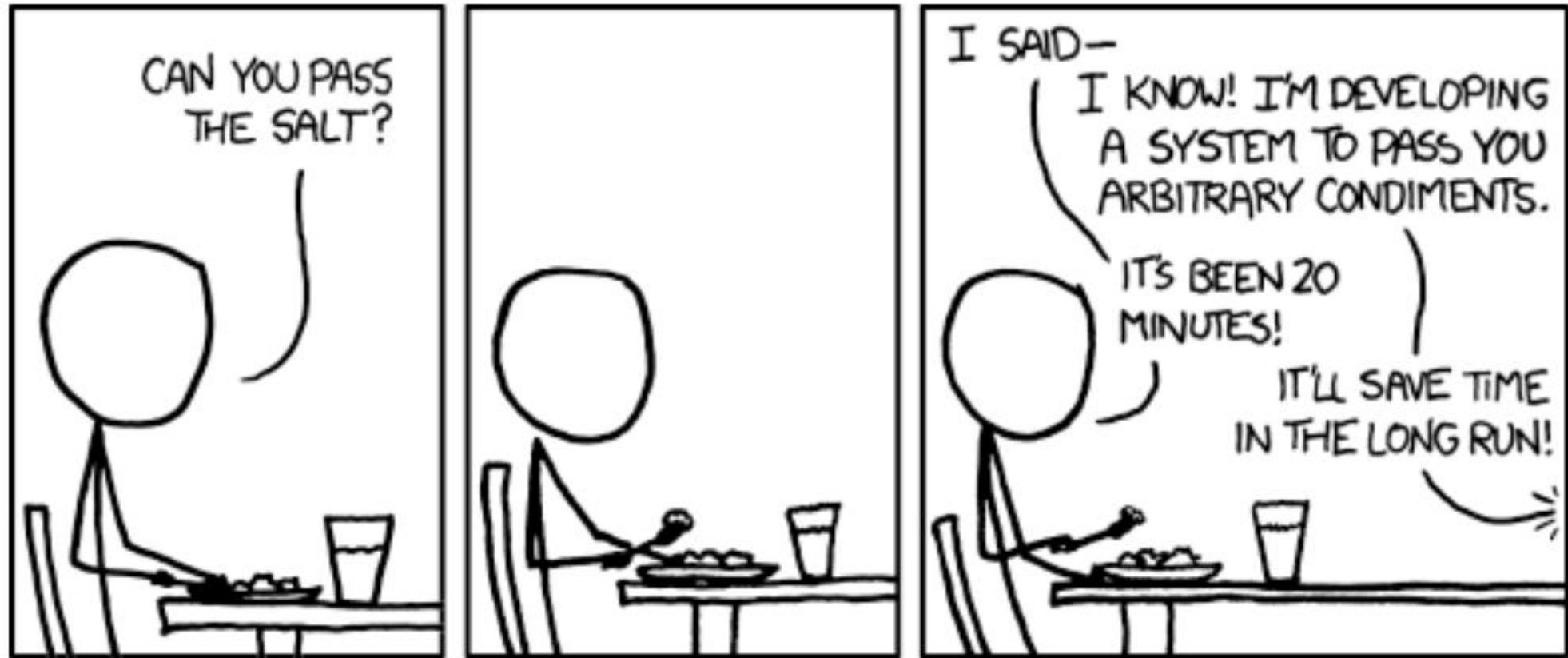


Fig. 59 Source: [XKCD #974](#)



HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE
EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

		HOW OFTEN YOU DO THE TASK					
		50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
HOW MUCH TIME YOU SHAVE OFF	1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
	5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
	30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
	1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
	5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
	30 MINUTES		6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS
	1 HOUR		10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS
	6 HOURS				2 MONTHS	2 WEEKS	1 DAY
	1 DAY					8 WEEKS	5 DAYS

Fig. 58 Source: XKCD #1205

○ Ulemper med optimalisering

- Tar tid fra andre ting (ny funksjonalitet, bedre testing, osv.)
- Kan gå ut over leseligheten (i noen tilfeller) - hva er viktigst?
- Kan resultere i bugs
- Vi må derfor veie fordelene med raskere kjøretid opp mot disse ulempene



○ Hvis vi skal optimalisere, hvordan gjør vi det?

- Analysere algoritmer for å finne den teoretiske forskjellen mellom ulike fremgangsmåter
- Ta tiden på koden (benchmarking) for å finne ut hvor rask den er i praksis
- Profilerings for å forstå hvilke deler av koden vi trenger å optimalisere



Benchmarking



- "Hvor nøyaktig er dette geværet?"
- Det varierte med hvor god skytteren (**marks**man) var med akkurat det våpenet, så for å eliminere den variabelen festet man våpen til en **benk** for å kunne sammenligne nøyaktigheten



Benchmarking



- I kodesammenheng er det viktig at man da tester ulike versjoner av koden mot *samme* benchmark (og ikke endrer test-caset over tid)
- Kan da teste ulike tilnærminger med samme input og se på kjøretid og nøyaktighet (på output)



LIVE CODING: PROFILING

