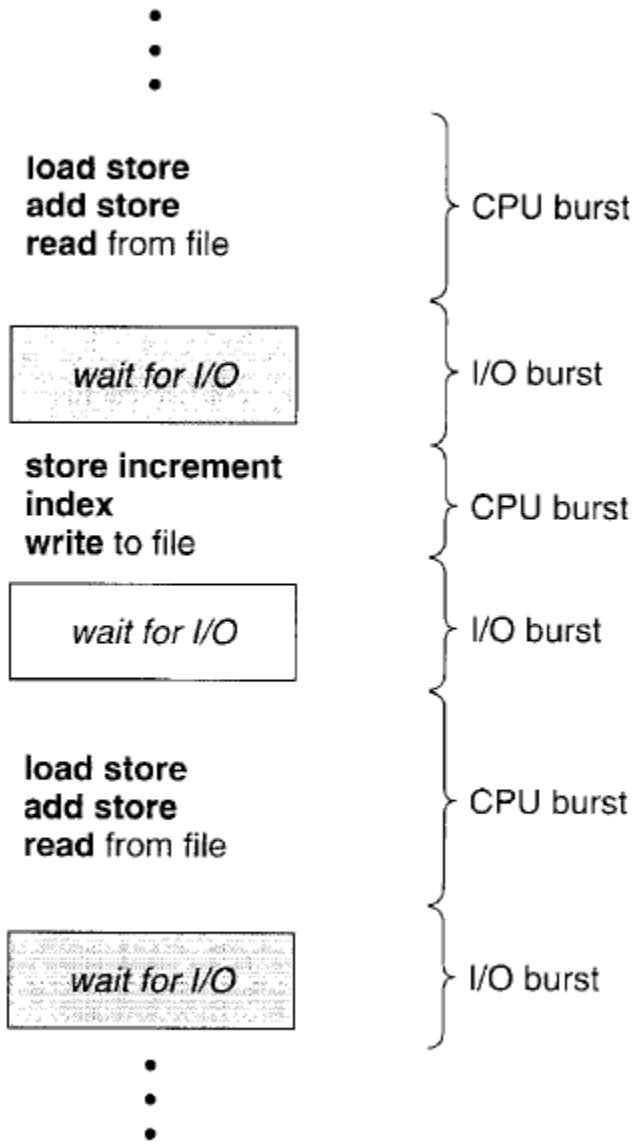
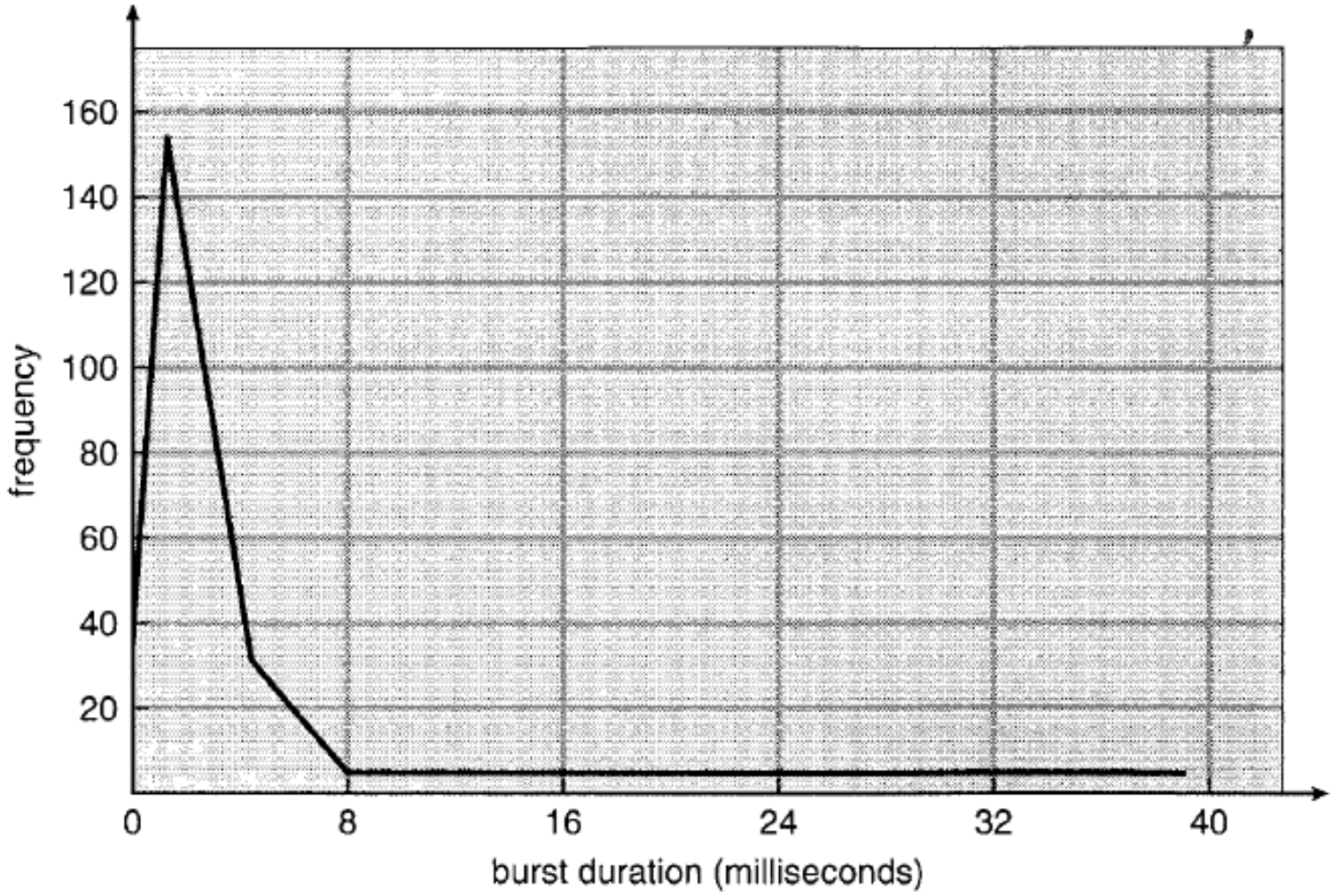
CPU and I/O bursts



CPU-burst Duration



Preemptive Scheduling

CPU-scheduling decision may take place under the following four circumstances.

1. **Running state -> waiting state (I/O request or waiting for a child)**
2. **Running state -> ready state (interrupt)**
3. **Waiting state -> ready state (I/O complete)**
4. **A process terminates.**

Case 1, 4: no choices, scheduling must take place.

Case 1: process ที่รันอยู่ต้องหยุดแน่นอน รันต่อไม่ได้ เพราะติด I/O wait

Case 4: process ที่รันอยู่ต้องหยุดแน่นอน รันต่อไม่ได้ เพราะ terminate แล้ว

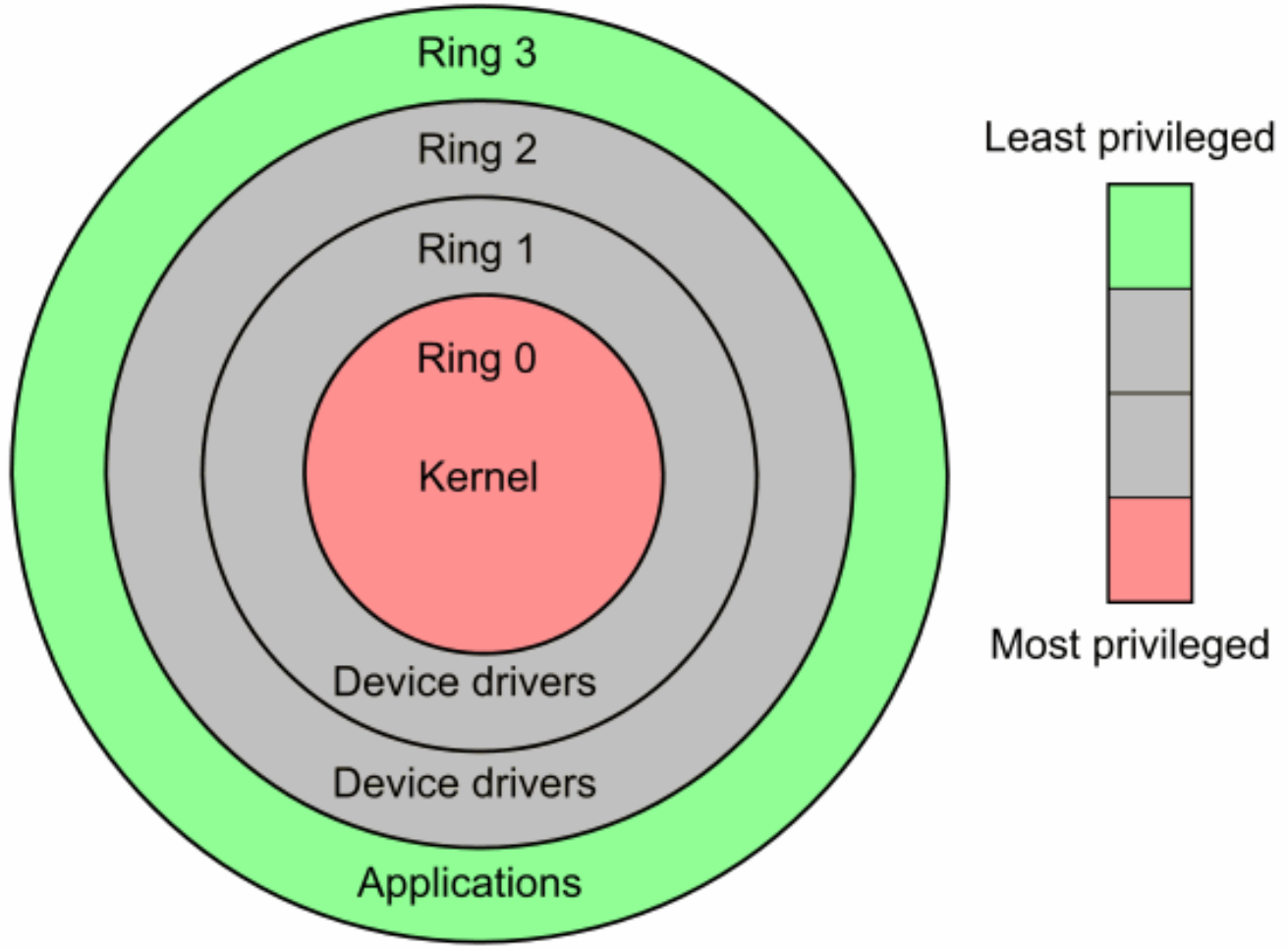
Case 2, 3: the running process has 2 choices, continue or break.

Case 2: process ที่รันอยู่ จะไม่หยุดก็ได้ หรือยอมให้ process ที่ interrupt ได้ใช้ CPU

Case 3: process ที่รันอยู่ จะไม่หยุดก็ได้ หรือยอมให้ process ที่ I/O complete ได้ใช้ CPU

ถ้าเกิด scheduling เฉพาะกรณี 1, 4 เรียกว่า nonpreemptive หรือ cooperative OS
ที่ไม่ยอมให้ preempt (ยึด CPU) เพราะ process ที่รันอยู่อาจจะยังทำ system call ยังไม่เสร็จ
ถ้าหยุดกลางทาง data structure จะพังได้ และยังไม่มีฮาร์ดแวร์ที่ช่วยป้องกัน critical section ด้วย (ดูบทต่อไป)
วิธีแก้ปัญหาในยุคแรกคือ ใช้ kernel ที่ทำ system call ให้เสร็จสมบูรณ์ก่อน แล้วค่อยคืน CPU

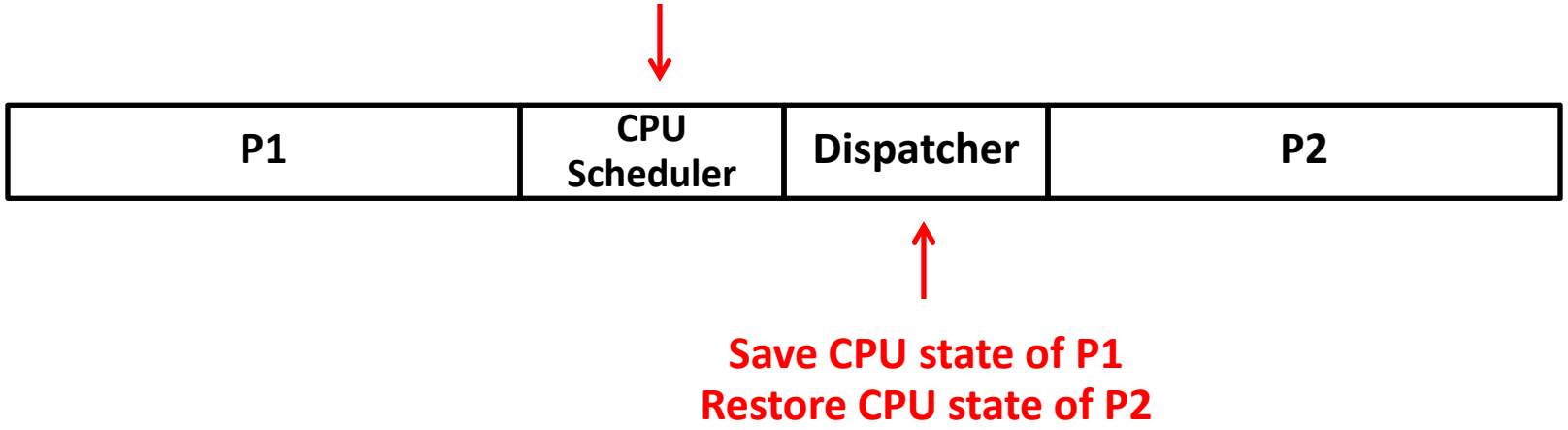
Kernel mode vs. User mode



Dispatcher

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program

Linux ใช้ SW ทำ CPU scheduling ในเวลา O(1) ไม่ขึ้นกับจำนวน process ใน ready queue



Hyperthreading: revisited

- ทั้ง **process** และ **thread** ต่างก็ใช้ **CPU scheduling** (มองเป็น **thread** ทั้งหมด)
- **Hyperthreading** จะหลอก **OS** ให้เห็นว่ามี **2 logical processor** ที่ทำงานเป็นอิสระจากกัน ทำให้ **CPU scheduler** สามารถใส่ **thread** ลงไปทำงานได้ 2 **thread** พร้อม ๆ กัน (แต่ไม่จำเป็นต้องใส่ลงไปพร้อมกันก็ได้) แต่ที่จริงแล้ว **2 logical processor** ไม่ได้ทำงานเป็นอิสระจากกัน แต่มี **physical processor** เดียวที่ทำทั้ง **2 thread** สลับไปมา

ในบทนี้ให้คิดว่า process คือ คำสั่งบน Linux เช่น ls เป็นต้น เป็น process ที่ทำเสร็จภายในเวลาสั้น ๆ ไม่ใช่ process ที่รันนานมาก และต้องรอผู้ใช้ terminate เอง เช่น word เป็นต้น

Linux เป็น OS แบบ multi-user มีผู้ใช้ remote login เข้ามาพร้อม ๆ กันได้ ดังนั้นจึงสนใจว่าจะใช้ Scheduling Algo. แบบใดจึงจะให้บริการได้ดีที่สุด

ต้องกำหนดเกณฑ์หรือ Scheduling Criteria ก่อน

Scheduling Criteria

ตัวชี้วัด **(benchmark)** ว่า **CPU Scheduler** ดีหรือไม่

1. **CPU Utilization** วัดจากช่วงเวลาหนึ่ง เช่น **10** วินาทีที่ผ่านมา
   = (เวลาที่ **CPU busy /** เวลาทั้งหมด**) x 100%**
2. **Throughput** ปริมาณงานที่ทำได้ วัดประสิทธิภาพ **(efficiency)**
   = จำนวน **process** ที่ทำเสร็จ ต่อ ระยะเวลา
   เช่น **10 process per second**
3. **Turnaround time** ในมุมมองของ **process** อยากให้ตัวมันเองเสร็จเร็ว ๆ
   = ค่าเฉลี่ยของเวลาที่ใช้ในการทำ **1 process** ให้เสร็จ
   นับเวลารวมทั้งหมดตั้งแต่เริ่มสร้าง **process** จนทำงานเสร็จ
4. **Waiting time (in ready queue)**
   = ค่าเฉลี่ยของเวลารวมทั้งหมดที่ **1 process** ต้องรอใน **ready queue**
5. **Response time** สำหรับ **interactive systems**
   The time from submission of a request until the first response is produced.

Scheduling Algorithms

1. First-come, First-served
2. Shortest-job-first
3. Priority
4. Round-Robin
5. Multilevel Queue
6. Multilevel Feedback Queue

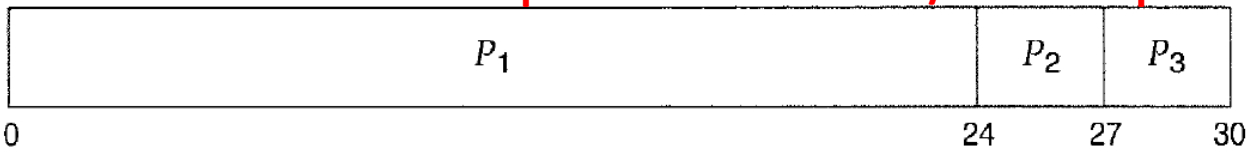
First-come, First-served

(non-preemptive)

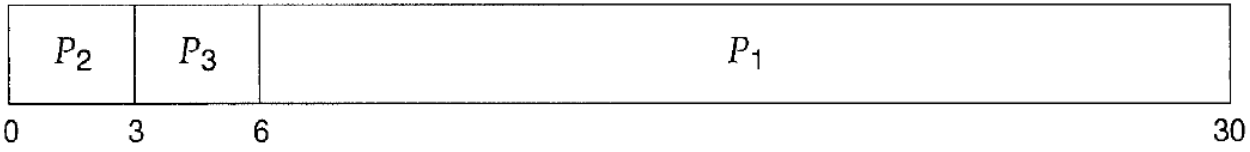
Process	Burst Time
P_1	24
P_2	3
P_3	3

If the processes arrive in the order P_1, P_2, P_3, and are served in FCFS order, we get the result shown in the following **Gantt chart**:

CPU-bound process **I/O-bound process**



The waiting time is 0 milliseconds for process P_1, 24 milliseconds for process P_2, and 27 milliseconds for process P_3. Thus, the average waiting time is $(0 + 24 + 27)/3 = 17$ milliseconds. If the processes arrive in the order P_2, P_3, P_1, however, the results will be as shown in the following Gantt chart:



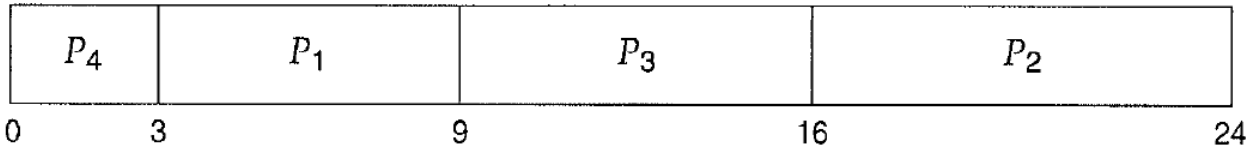
The average waiting time is now $(6 + 0 + 3)/3 = 3$ milliseconds. This reduction is substantial. Thus, the average waiting time under an FCFS policy is generally not minimal and may vary substantially if the process's CPU burst times vary greatly.

Convoy effect

- All the other processes wait for the one big process to get off the CPU (increasing waiting time).

Shortest-job-first

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3



Provably optimal waiting time.

The next CPU burst is generally predicted as an exponential average of the measured lengths of previous CPU bursts. Let t_n be the length of the nth CPU burst, and let τ_{n+1} be our predicted value for the next CPU burst. Then, for α, $0 \leq \alpha \leq 1$, define

Weight ว่าจะเชื่ออันไหนมากกว่า

(Tau) $\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n.$

Estimated burst ครั้งที่ n+1

Actual burst ครั้งที่ n

Estimated burst ครั้งที่ n

Weight ว่าจะเชื่ออันไหนมากกว่า

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n.$$

Estimated burst ครั้งที่ n+1

Actual burst ครั้งที่ n

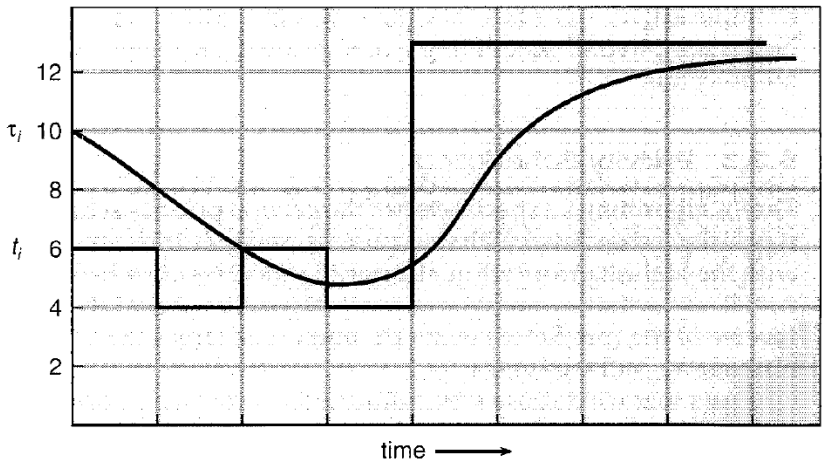
Estimated burst ครั้งที่ n

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \cdots + (1 - \alpha)^j \alpha t_{n-j} + \cdots + (1 - \alpha)^{n+1} \tau_0.$$

Actual burst time ในอดีต และ weight น้อยลงเรื่อยๆ เป็น exponential

Estimated (ไม่ continuous)

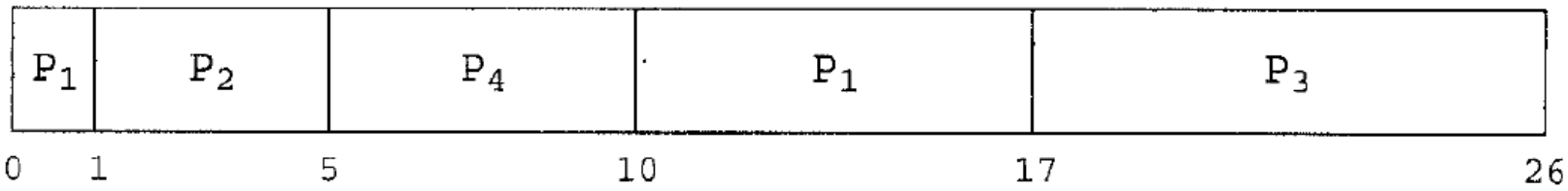
Actual



CPU burst (t_i)	6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

สมมติว่ารู้

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



แบบ preemptive (จะได้ waiting time น้อยกว่า)

The average waiting time for this example is $((10 - 1) + (1 - 1) + (17 - 2) + (5 - 3))/4 = 26/4 = 6.5$ milliseconds. Nonpreemptive SJF scheduling would result in an average waiting time of 7.75 milliseconds.

ลองคิดดูว่าทำไมแบบ non-preemptive ได้ 7.75

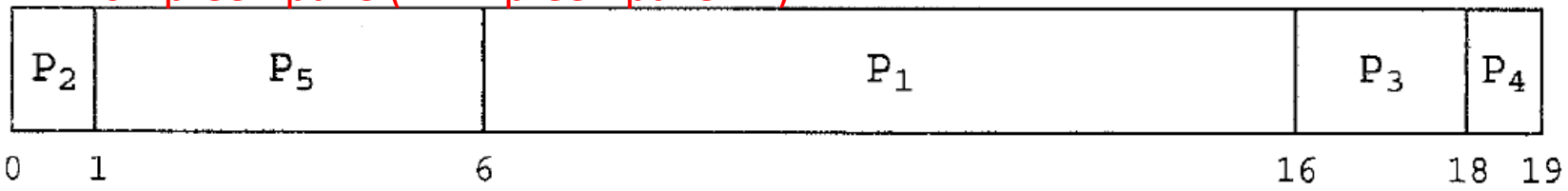
Priority

มาพร้อมกัน
หมดทีเดียว

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

ค่าน้อย priority มาก

แบบ non-preemptive (ทำแบบ preemptive ก็ได้)



Problem

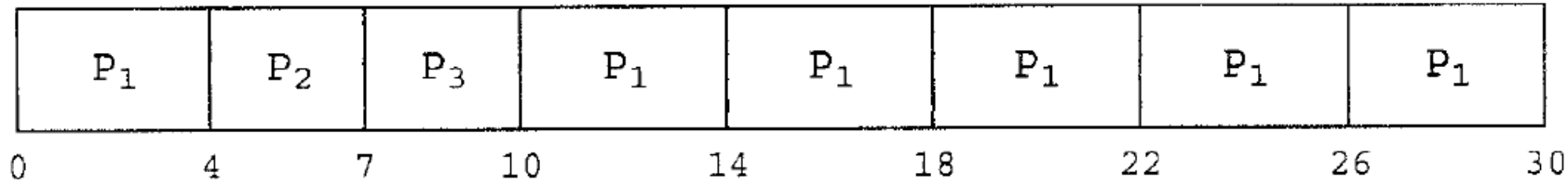
Indefinite blocking or starvation

แก้ด้วย aging (เพิ่ม priority ของ process ทุก ๆ 1 นาที)

Round-Robin

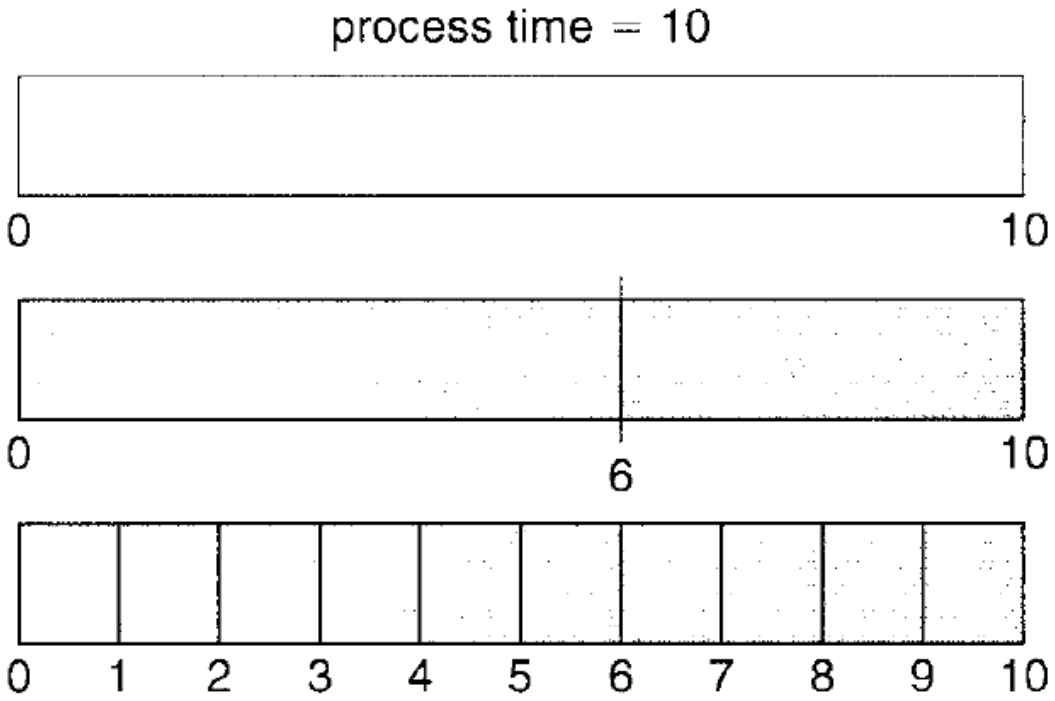
Process	Burst Time
P_1	24
P_2	3
P_3	3

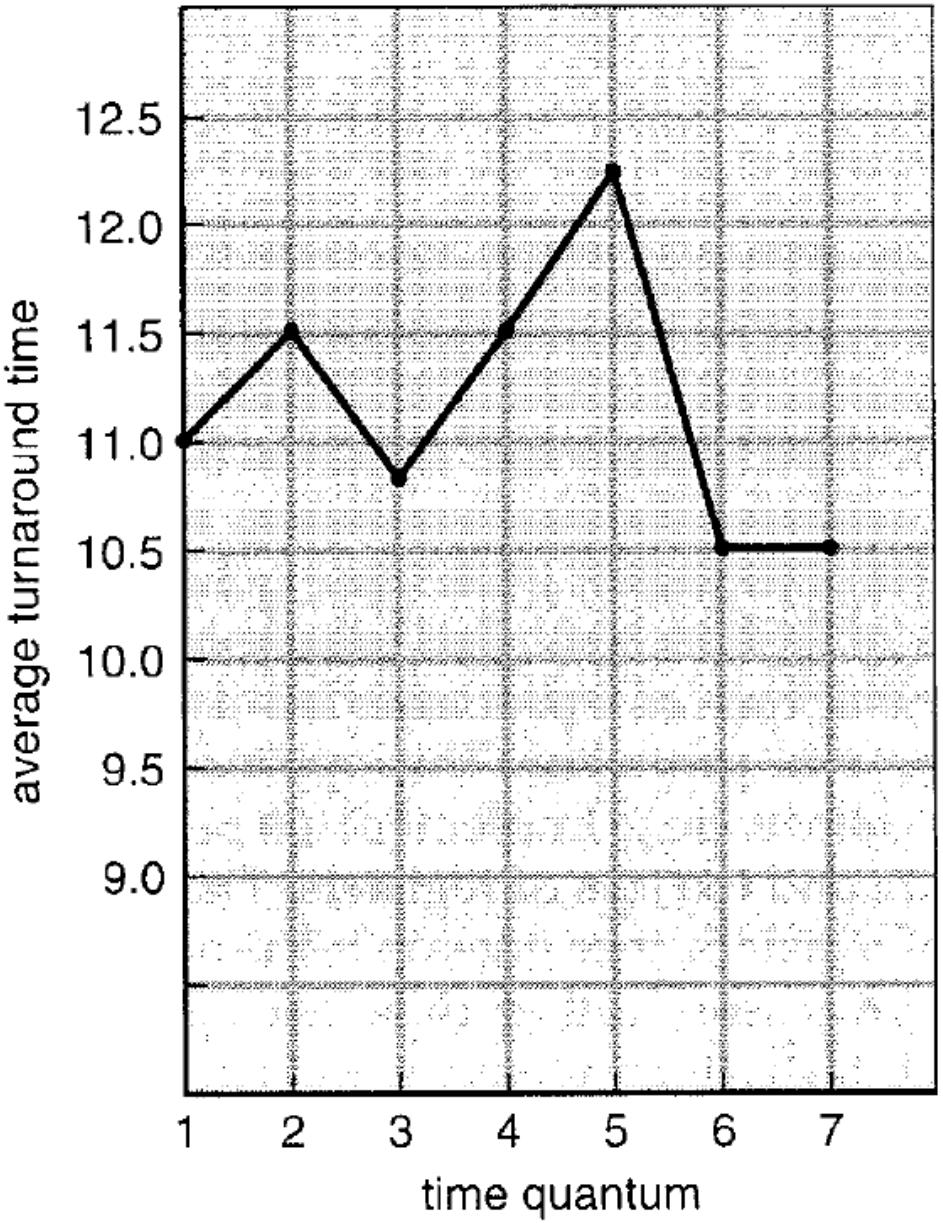
Time slice = 4



The average waiting time is 17/3 = 5.66 milliseconds.

Thus, we want the time quantum to be large with respect to the context-switch time. If the context-switch time is approximately 10 percent of the time quantum, then about 10 percent of the CPU time will be spent in context switching. In practice, most modern systems have time quanta ranging from 10 to 100 milliseconds. The time required for a context switch is typically less than 10 microseconds; thus, the context-switch time is a small fraction of the time quantum.





process	time
P_1	6
P_2	3
P_3	1
P_4	7

เกิด context switch บ่อย
Turnaround time มาก

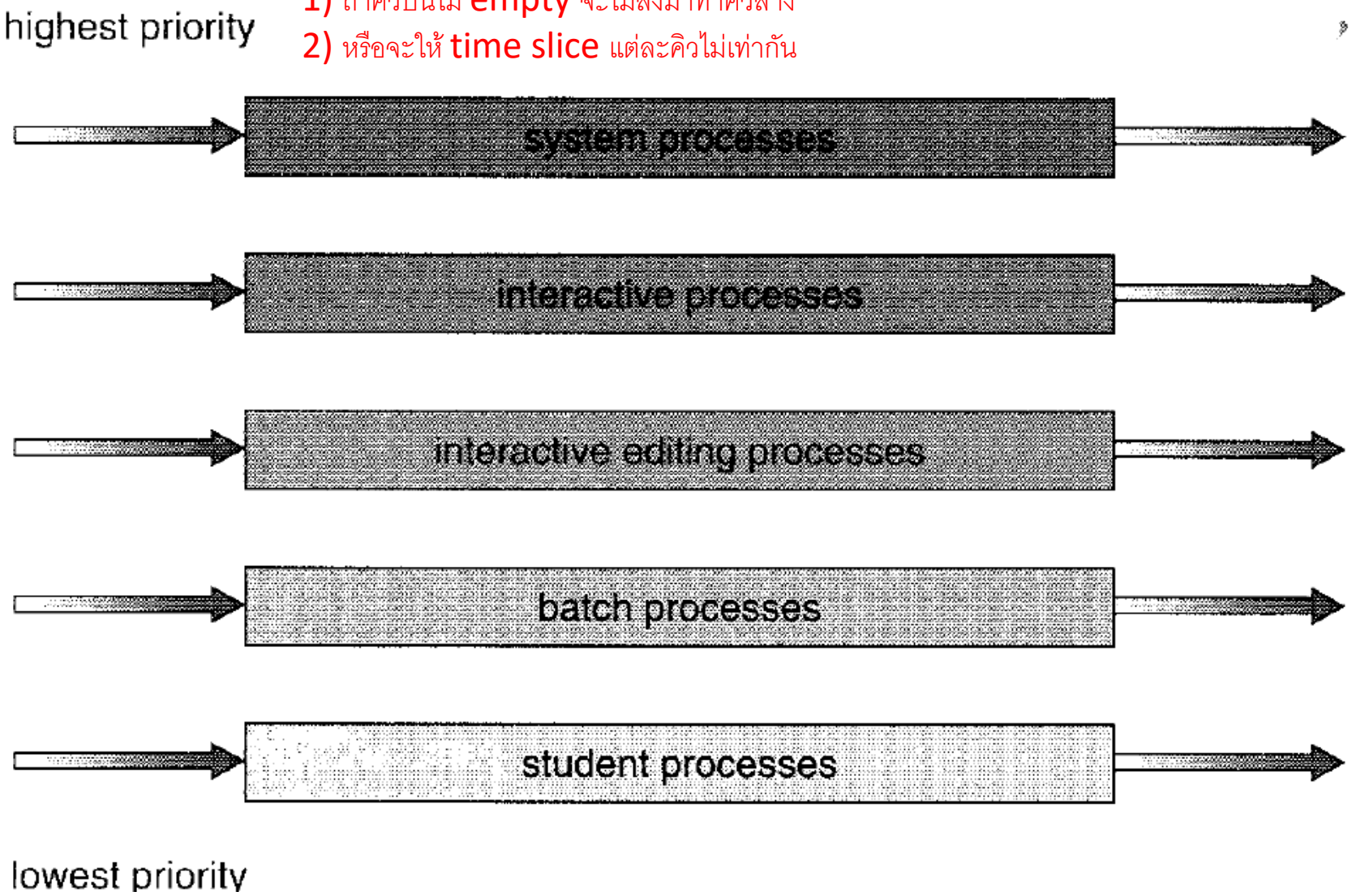
ไม่เป็น
time sharing

น้อย ← → มาก

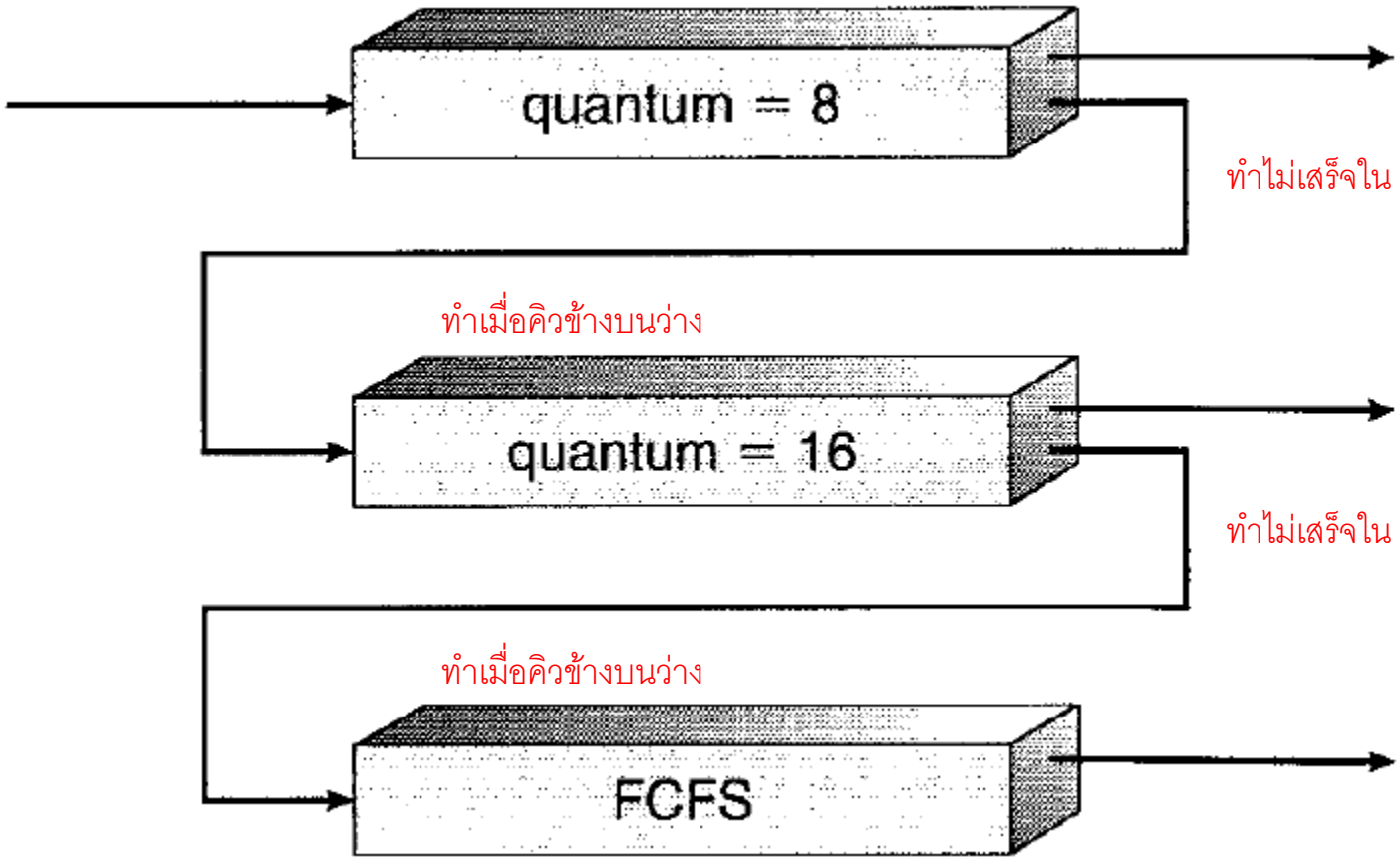
Time quantum
(time slice)

Multilevel Queue

1) ถ้าคิวบนไม่ empty จะไม่ลงมาทำคิวล่าง
2) หรือจะให้ time slice แต่ละคิวไม่เท่ากัน



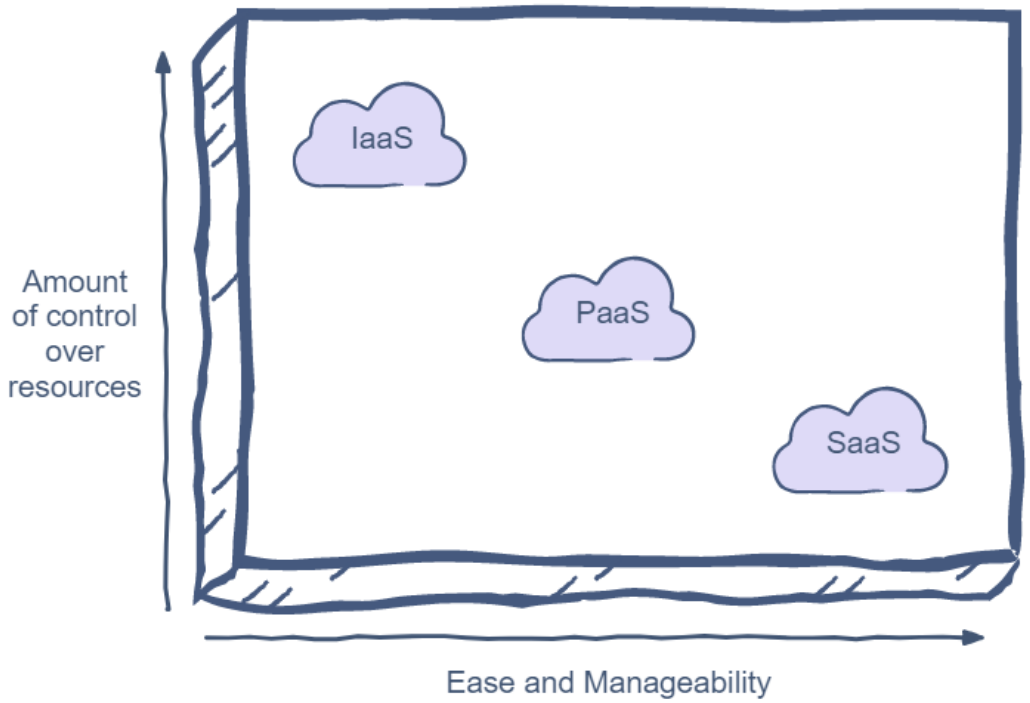
Multilevel Feedback Queue



รันไปให้หมด CPU burst เลย
เป็น CPU-bound process
ไม่ต้องทำ time slice แล้ว ไม่คุ้ม

Cloud Computing

- Infrastructure as a service (IaaS)
- Platform as a service (PaaS)
- Software as a service (SaaS)



Virtual Machines
(Windows, Linux)

Software Development
เช่น https://www.heroku.com

Google Apps
Microsoft Office 365
Gmail, Facebook

อาเซ่อ (US) / อาซัว (EU)

Fixed-Size XS

฿150

MONTHLY

- CPU 1 Core
- RAM 2 GB
- SSD 30 GB
- 1 Gbps Shared
- Unlimited Data Transfers

Buy Now

Fixed-Size R1

฿250

MONTHLY

- CPU 1 Core
- RAM 3 GB
- SSD 60 GB
- 1 Gbps Shared
- Unlimited Data Transfers

Buy Now

Fixed-Size R2

฿500

MONTHLY

- CPU 2 Cores
- RAM 6 GB
- SSD 120 GB
- 1 Gbps Shared
- Unlimited Data Transfers

Buy Now

Fixed-Size R3

฿1,000

MONTHLY

- CPU 4 Cores
- RAM 12 GB
- SSD 240 GB
- 1 Gbps Shared
- Unlimited Data Transfers

Buy Now

Fixed-Size R4

฿1,500

MONTHLY

- CPU 5 Cores
- RAM 18 GB
- SSD 360 GB
- 1 Gbps Shared
- Unlimited Data Transfers

Buy Now

1 CPU [core]

1 core = ฿100

1

2 RAM [MB]

1024MB = ฿20

1024

3 SSD [GB]

1GB = ฿2

20

Change billing cycle

160.00 THB Monthly

Total Recurring:

160.00 THB Monthly

Order

aws

ติดต่อฝ่ายขาย | สนับสนุน | ไทย | บัญชีของฉัน | ลงชื่อเข้าใช้คอนโซล

ผลิตภัณฑ์ | โซลูชัน | ราคา | เอกสาร | เรียนรู้ | เครือข่ายคู่ค้า | AWS Marketplace | การพัฒนาลูกค้า | สำรวจเพิ่มเติม

AWS Management Console | ภาพรวม | คุณสมบัติ | คำถามที่พบบ่อย

AWS Management Console

ทุกสิ่งที่คุณต้องใช้ในการเข้าถึงและจัดการ AWS Cloud ในอินเตอร์เฟสเว็บเดียว

สร้างบัญชีฟรี

มีบัญชีอยู่แล้วใช่ไหม ลงชื่อเข้าใช้

AWS Management Console * จะมอบความกว้างและความลึกของ AWS ที่เหนือใครมาไว้ในคอมพิวเตอร์หรือโทรศัพท์มือถือของคุณด้วยพอร์ทัลบนเว็บที่ปลอดภัยและเข้าถึงได้ง่าย ค้นพบบริการใหม่ๆ จัดการทั้งบัญชีของคุณ สร้างแอปพลิเคชันใหม่ และเรียนรู้วิธีการทำงานที่มากขึ้นกว่าเดิมด้วย AWS

ภาพรวมของ Console

- ค้นพบและทดลองใช้บริการของ AWS กว่า 150 รายการซึ่งมีให้ทดลองใช้ฟรีมากมาย
- สร้างแอปพลิเคชันบนระบบคลาวด์ของคุณในศูนย์ข้อมูล AWS ใดก็ได้ทั่วโลก
- จัดการและตรวจสอบผู้ใช้ การใช้บริการสภาพ และการเก็บค่าบริการรายเดือน
- รับความช่วยเหลือใน Console จาก AWS Support

สร้างบัญชีฟรี

https://aws.amazon.com/th/console/

Compute
EC2
Lightsail
ECR
ECS
EKS
Lambda
Batch
Elastic Beanstalk
Serverless Application Repository

Storage
S3
EFS
FSx
S3 Glacier
Storage Gateway
AWS Backup

Database
RDS
DynamoDB
ElastiCache
Neptune
Amazon Redshift
Amazon QLDB
Amazon DocumentDB

Robotics
AWS RoboMaker

Blockchain
Amazon Managed Blockchain

Satellite
Ground Station

Management & Governance
AWS Organizations
CloudWatch
AWS Auto Scaling
CloudFormation
CloudTrail
Config
OpsWorks
Service Catalog
Systems Manager
Trusted Advisor
Managed Services
Control Tower
AWS License Manager
AWS Well-Architected Tool
Personal Health Dashboard
AWS Chatbot

Analytics
Athena
EMR
CloudSearch
Elasticsearch Service
Kinesis
QuickSight
Data Pipeline
AWS Glue
AWS Lake Formation
MSK

Security, Identity, & Compliance
IAM
Resource Access Manager
Cognito
Secrets Manager
GuardDuty
Inspector
Amazon Macie
AWS Single Sign-On
Certificate Manager
Key Management Service
CloudHSM
Directory Service
WAF & Shield
Artifact
Security Hub

Business Applications
Alexa for Business
Amazon Chime
WorkMail

End User Computing
WorkSpaces
AppStream 2.0
WorkDocs
WorkLink

Internet Of Things
IoT Core
Amazon FreeRTOS
IoT 1-Click
IoT Analytics
IoT Device Defender
IoT Device Management
IoT Events
IoT Greengrass
IoT SiteWise
IoT Things Graph

Game Development
Amazon GameLift

50 ALEXA COMMANDS

https://www.youtube.com/watch?v=xenOYWVwkGY

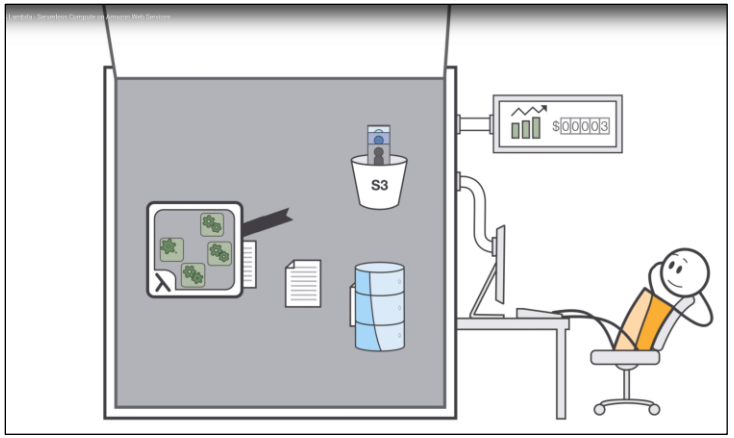
AWS Lambda

เรียกใช้โค้ดโดยไม่ต้องคำนึงถึงเซิร์ฟเวอร์ จ่ายเฉพาะเวลาการประมวลผลที่คุณใช้

เริ่มต้นใช้งาน AWS Lambda

เหมาะกับงานที่ต้องการให้ต้นทุนแปรตามคำสั่งซื้อ

ไม่ fix cost ของต้นทุนไว้



https://youtu.be/eOBq__h4OJ4