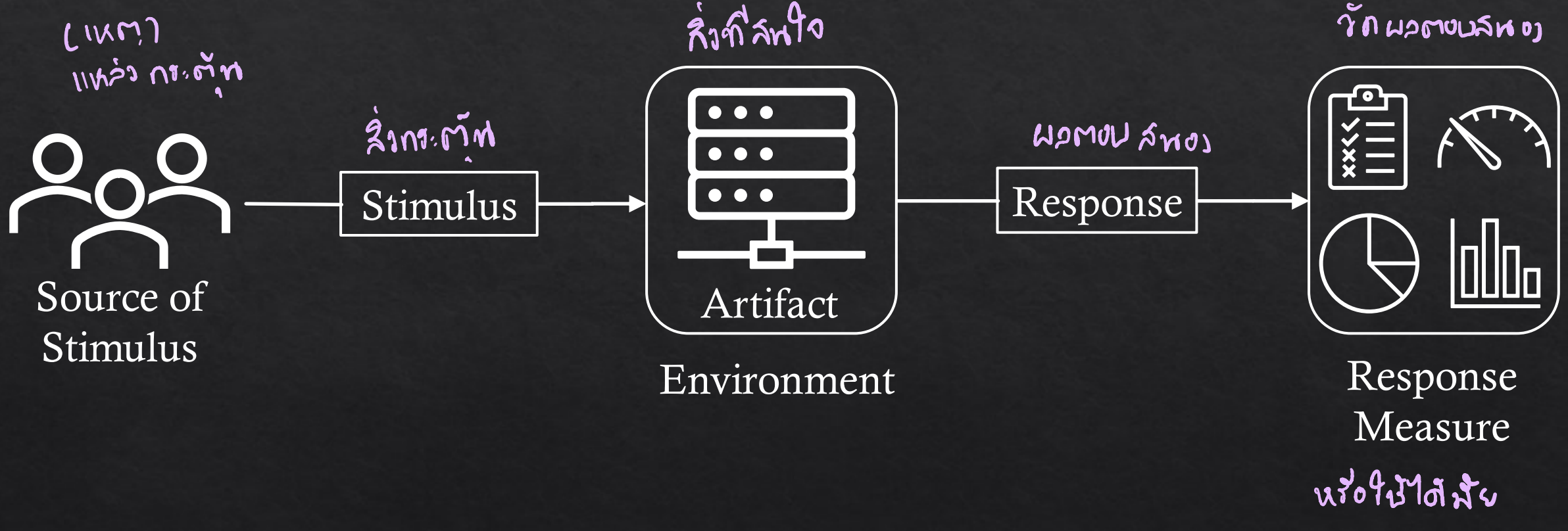
Quality Attributes

Parinya Ekparinya

Parinya.Ek@kmitl.ac.th

Quality Attribute Scenario



ความสามารถของระบบรองรับการเปลี่ยนแปลง

MODIFIABILITY

Modifiability

- A system's modifiability refers to its receptiveness to change.

  McGovern, J., Tyagi, S., Stevens, M., & Mathew, S. (2003). Java web services architecture. Elsevier.

- Change happens: แก้ตรงจาก เพิ่มฟีเจอ เปลี่ยน ฟีเจอเก่าออก
  - to add new features, to change or even retire old ones. เพิ่ม feature , เปลี่ยน / ปรับปรุงของเก่า
  - to fix defects, tighten security, or improve performance. แก้ไขข้อบกพร่อง , กระชับความปลอดภัย , หรือ ปรับปรุงประสิทธิภาพ
  - to enhance the user's experience. Changes happen to embrace new technology, new platforms, new protocols, new standards. เพิ่มประสบการณ์ผู้ใช้ . เปลี่ยนแปลงเพื่อรองรับ เทคโนโลยีใหม่ แพลตฟอร์ม โปรโตคอลใหม่
  - to make systems work together, even if they were never designed to do so. ทำให้ระบบทำงานร่วมกันได้

ออกแบบให้ Sys ทำงานร่วมกัน แม้ ไม่ได้ถูกออกแบบมาให้ทำ

What is the cost of the change?

- Making a system more modifiable involves two types of costs:
  - The cost of introducing the mechanism(s) to make the system more modifiable
  - The cost of making the modification using the mechanism(s)
- For *N* similar modifications, a simplified justification for a change mechanism is that:

$$\begin{aligned} &N * \textit{Cost of making change without the mechanism} \\ &\leq \\ &\textit{Cost of creating the mechanism} \\ &+ \\ &(N * \textit{cost of making the change using the mechanism}) \end{aligned}$$

Modifiability General Scenario (1/2)

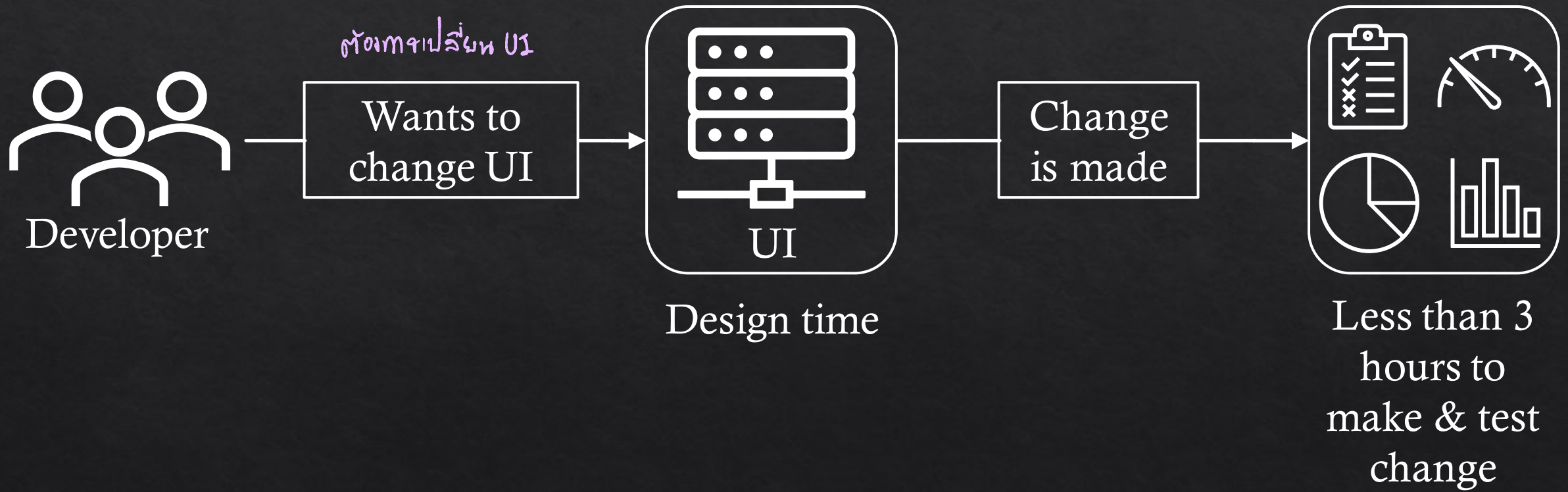
ใครอยากเปลี่ยน (เหตุกระตุ้น)

Source of stimulus	End user, developer, system administrator, product line owner, the system itself
Stimulus	A directive to add/delete/modify functionality, or change a quality attribute, capacity, platform, or technology; A directive to add new product line; A directive to change the location of a service to another location
Artifacts สิ่งที่เปลี่ยน (ที่สนใจ)	Code, data, interfaces, components, resources, configurations, documentation, …
Environment เปลี่ยนตอนแวดล้อม	Runtime, compile time, build time, initiation time, design time services เริ่ม / Design time

Modifiability General Scenario (2/2)

Response	One or more of the following: • Make modification • Test modification • Deploy modification • Self-modify ให้ระบบ modify ตัวเองได้
Response measure	Cost in terms of the following: • Number, size, complexity of affected artifacts • Effort แรง (เวลาทำงาน/โปรแกรม) • Elapsed time • Money (direct outlay or opportunity cost) • Extent to which this modification affects other functions or quality attributes • New defects introduced • How long it took the system to adapt

Sample Modifiability Scenario



Tactics for Modifiability

Increase Cohesion	Reduce Coupling	Defer Binding
• Split module • Redistribute responsibilities	• Encapsulate • Use an intermediary • Abstract common services • Restrict dependencies	• Component replacement • Compile-time parameterization • Aspects • Configuration-time binding • Resource files • Discovery • Interpret parameters • Shared repositories • Polymorphism

ห้อย - กระจาย
มาก - กลุ่มก้อน

Cohesion

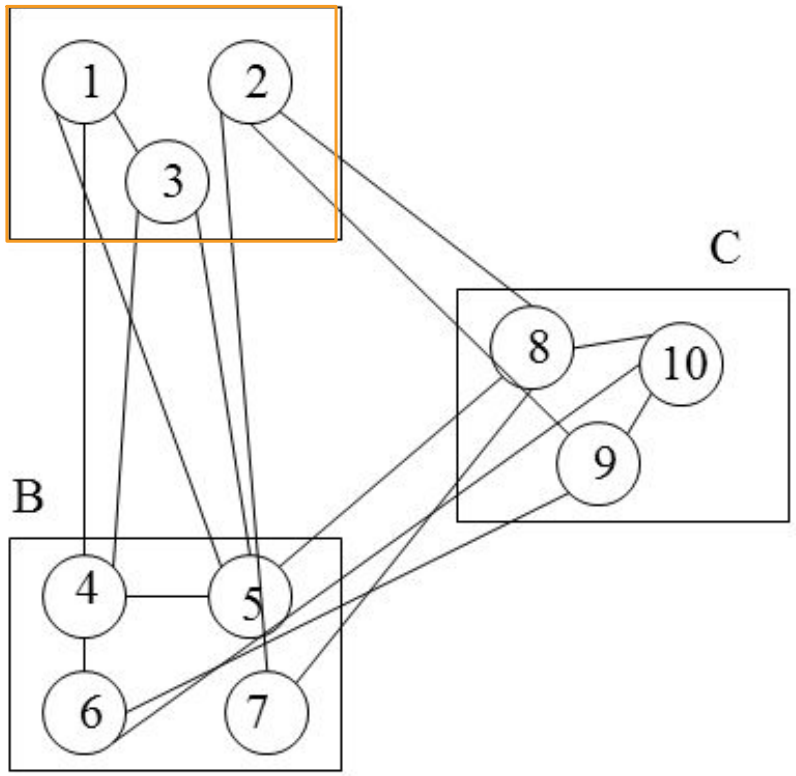
เกี่ยวข้อง สัมพันธ์มาก แค่ไหน

- Cohesion measures how strongly the responsibilities of a module are related.
- The cohesion of a module is the probability that a change scenario that affects a responsibility will also affect other (different) responsibilities.
- The higher the cohesion, the lower the probability that a given change will affect multiple responsibilities.
- If module A has a low cohesion, then cohesion can be improved by removing responsibilities unaffected by anticipated changes.

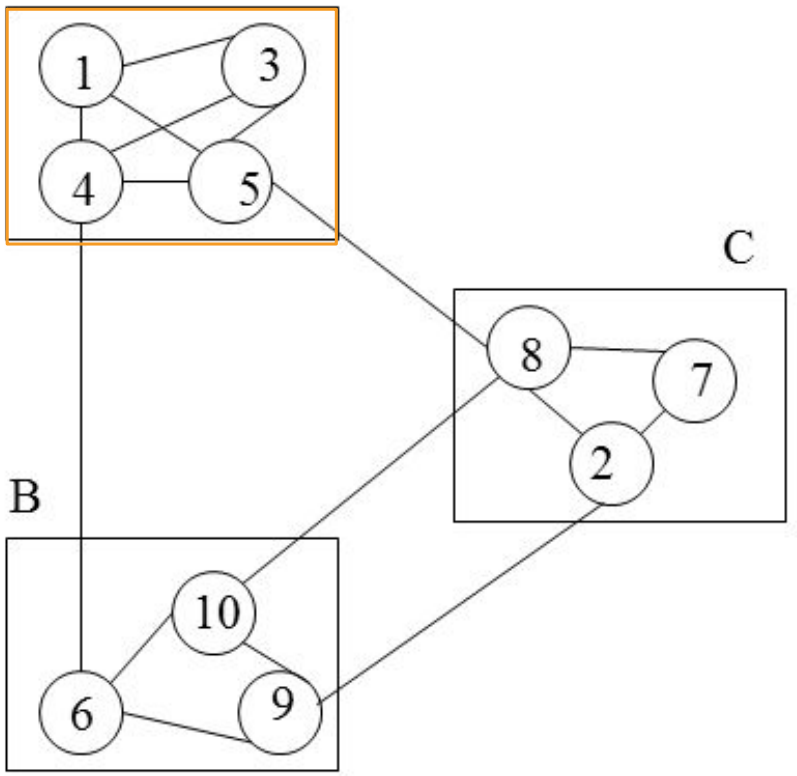
เปลี่ยนหาสิ่ง ผลกับให้ module อื่นๆเกี่ยวข้องต้องเปลี่ยน

Coupling

- Modules have responsibilities. When a change causes a module to be modified, its responsibilities are changed in some way.
- Generally, a change that affects one module is easier and less expensive than if it changes more than one module.
- However, if two modules' responsibilities overlap in some way, then a single change may well affect them both.
- We can measure this overlap by measuring the probability that a modification to one module will propagate to the other. This is called coupling, and high coupling is an enemy of modifiability.



Bad modularization:
low cohesion, high coupling



Good modularization:
high cohesion, low coupling

Source: https://vivadifferences.com/difference-between-coupling-and-cohesion-in-software-engineering-with-examples/

Increase Cohesion

แยกย่อยสิ่งที่เปลี่ยงปัน

- **Split Module**: If the module being modified includes a great deal of capability, the modification costs will likely be high. Refining the module into several smaller modules should reduce the average cost of future changes.
- **Redistribute responsibilities**: If responsibilities A, A′, and A″ (all similar responsibilities) are sprinkled across several distinct modules, they should be placed together. This refactoring may involve creating a new module, or it may involve moving responsibilities to existing modules.

จัดกลุ่มเดียวกันใหม่

Reduce Coupling

- **Encapsulate**: Encapsulation introduces an explicit interface to a module. This interface includes an API and its associated responsibilities, such as "perform a syntactic transformation on an input parameter to an internal representation."
- **Use an Intermediary**: Given a dependency between responsibility A and responsibility B (for example, carrying out A first requires carrying out B), the dependency can be broken by using an intermediary.
- **Abstract Common Services**: where two modules provide not-quite-the-same but similar services, it may be cost-effective to implement the services just once in a more general (abstract) form.
- **Restrict Dependencies**: restricts the modules which a given module interacts with or depends on.

Defer Binding (1/2)

- In general, the later in the life cycle we can bind values, the better. If we design artifacts with built-in flexibility, then exercising that flexibility is usually cheaper than hand-coding a specific change.
- However, putting the mechanisms in place to facilitate that late binding tends to be more expensive.
- The equation given earlier comes into play. We want to bind as late as possible, as long as the mechanism that allows it is cost-effective.
- The following tactics can be used to bind values at compile time or build time:
  - Component replacement (for example, in a build script or makefile)
  - Compile-time parameterization
  - Aspects

Defer Binding (2/2)

- The following tactics are available to bind values at deployment, startup time, or initialization time:
  - Configuration-time binding
  - Resource files
- Tactics to bind values at runtime include the following:
  - Discovery
  - Interpret parameters
  - Shared repositories
  - Polymorphism

PERFORMANCE

Performance

- Performance measures how effective is a software system with respect to time constraints and allocation of resources.

Cortellessa V., Di Marco A., Inverardi P. (2011) What Is Software Performance?. In: Model-Based Software Performance Analysis. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-13621-4_1

Performance General Scenario (1/3)

Source of stimulus	External • User request • Request from external system • Data arriving from a sensor or other system Internal • One component may make a request of another component. • A timer may generate notification.
Stimulus	Arrival of a periodic, sporadic, or stochastic event: • A periodic event arrives at a predictable interval. • A stochastic event arrives according to some probability distribution. • A sporadic event arrives according to a pattern that is neither periodic nor stochastic.

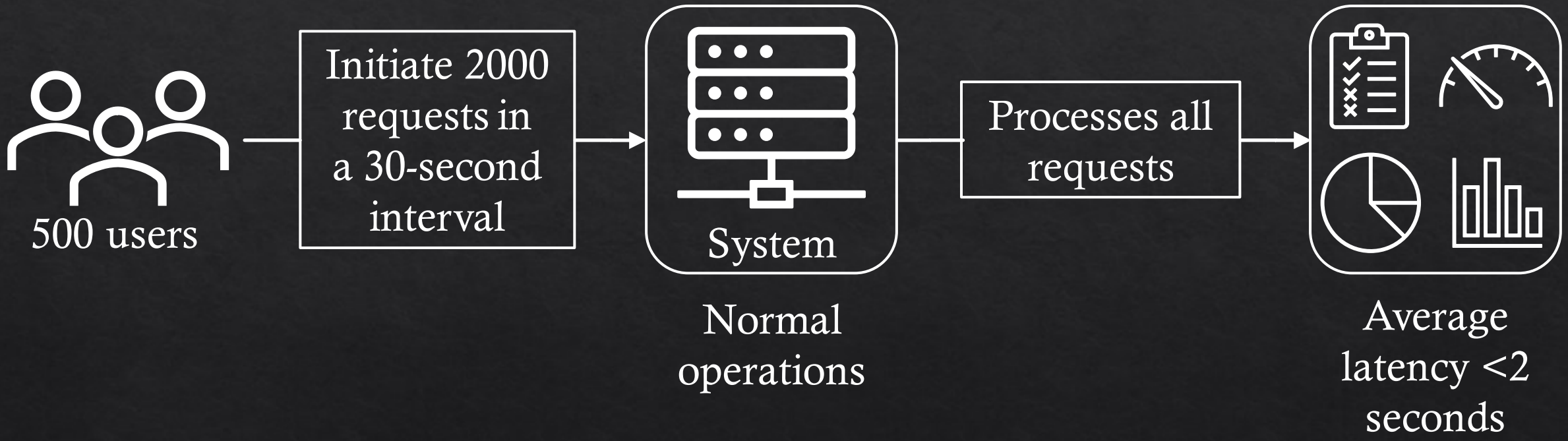
Performance General Scenario (2/3)

Artifacts	• Whole system • Component within system
Environment	Runtime. The system or component can be operating in: • Normal mode • Emergency mode • Error correction mode • Peak load • Overload • Degraded operation mode • Some other defined mode of the system

Performance General Scenario (3/3)

Response	• System returns a response • System returns an error • System generates no response • System ignores the request if overloaded • System changes the mode or level of service • System services a higher-priority event • System consumes resources
Response measure	• The (maximum, minimum, mean, median) time the response takes (latency) • The number or percentage of satisfied requests over some time interval (throughput) or set of events received • The number or percentage of requests that go unsatisfied • The variation in response time (jitter) • Usage level of a computing resource

Sample Performance Scenario



Tactics for Performance

Control Resource Demand	Manage Resources
• Manage work requests • Limit event response • Prioritize events • Reduce computational overhead • Bound execution times • Increase efficiency	• Increase resources • Introduce concurrency • Maintain multiple copies of computations • Maintain multiple copies of data • Bound queue sizes • Schedule resources

Control Resource Demand

Manage Work Requests

- **Manage Event Arrival:** A common way to manage event arrivals from an external system is to put in place a service level agreement (SLA), an agreement of the form "The system or component will process X events arriving per unit time with a response time of Y."
- **Manage Sampling Rate:** In cases where the system cannot maintain adequate response levels, you can reduce the sampling frequency of the stimuli—for example, the rate at which data is received from a sensor or the number of video frames per second that you process. The tradeoff is the fidelity of the video stream or the information you gather from the sensor data.

จำกัด จ.น. Response

Limit Event Response

- When discrete events arrive at the system (or component) too rapidly to be processed, then the events must be queued until they can be processed, or they are simply discarded.
- You may choose to process events only up to a set maximum rate, thereby ensuring predictable processing for the events that are actually processed.
- This tactic could be triggered by a queue size or processor utilization exceeding some warning level. Alternatively, it could be triggered by an event rate that violates an SLA.
- If you adopt this tactic and it is unacceptable to lose any events, then you must ensure that your queues are large enough to handle the worst case.

Prioritize Events

จัดลำดับความสำคัญ

- If not all events are equally important, you can impose a priority scheme that ranks events according to how important it is to service them.
- If insufficient resources are available to service them when they arise, low-priority events might be ignored.
- For example, a building management system may raise a variety of alarms. Life-threatening alarms such as a fire alarm should be given higher priority than informational alarms such as a room being too cold.

Reduce Computational Overhead

- **Reduce Indirection**: The use of intermediaries increases the computational overhead in processing an event stream, so removing them improves latency. This is a classic modifiability/performance tradeoff. Rather than a single component, a chain of components can also increase the processing overhead necessary to service an event.
- **Co-locate Communicating Resources**: Context switching and intercomponent communication costs add up, especially when the components are on different nodes on a network. Co-location may mean hosting cooperating components on the same processor to avoid the time delay of network communication.
- **Periodic Cleaning**: A special case when reducing computational overhead is to perform a periodic cleanup of resources that have become inefficient. For example, hash tables and virtual memory maps may require recalculation and reinitialization. Many system administrators do a periodic reboot of their systems for exactly this reason.

ขีดขอบเขตเวลาที่ Response

Bound Execution Times

- You can place a limit on how much execution time is used to respond to an event.
- For iterative, data-dependent algorithms, limiting the number of iterations is a method for bounding execution times.
- The cost, however, is usually a less accurate computation.

- Response ในระยะไม่เกินเท่าไหร่
- เกินแล้วจะทำไง

Increase Efficiency of Resource Usage

ฟังชั่น อัลกอริทึม อาจมีประสิทธิภาพดีขึ้น

- Improving the efficiency of algorithms used in critical areas can decrease latency and improve throughput and resource consumption.
- This is, for some programmers, their primary performance tactic. If the system does not perform adequately, they try to "tune up" their processing logic.

จัดการทรัพยากร

Manage Resources

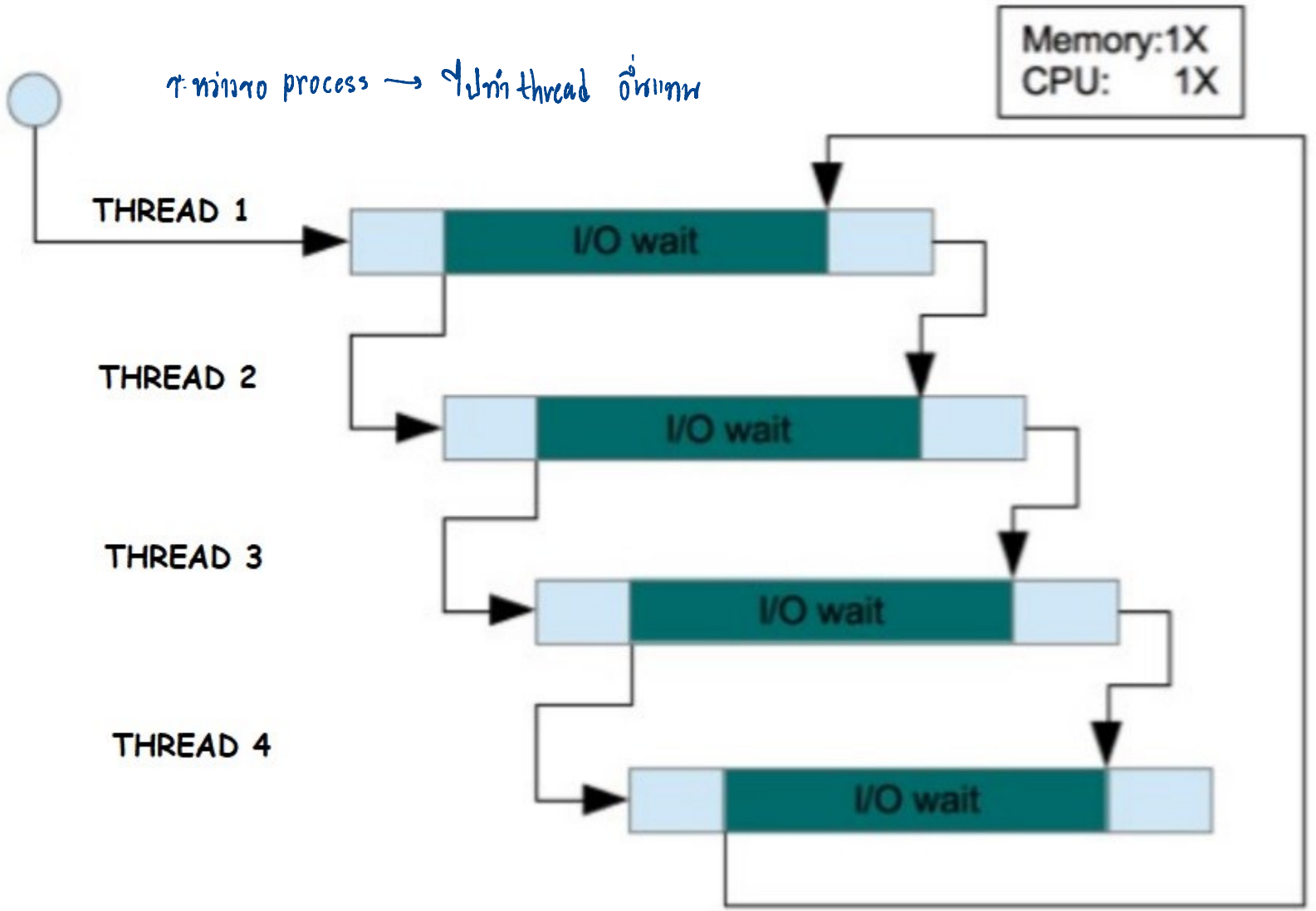
Increase Resources

เพิ่ม ทรัพยากร (เพิ่ม RAM, เพิ่ม CPU, หรือ network เร็ว)

- Faster processors, additional processors, additional memory, and faster networks all have the potential to improve performance.
- Cost is usually a consideration in the choice of resources, but increasing the resources is, in many cases, the cheapest way to get immediate improvement.

Introduce Concurrency

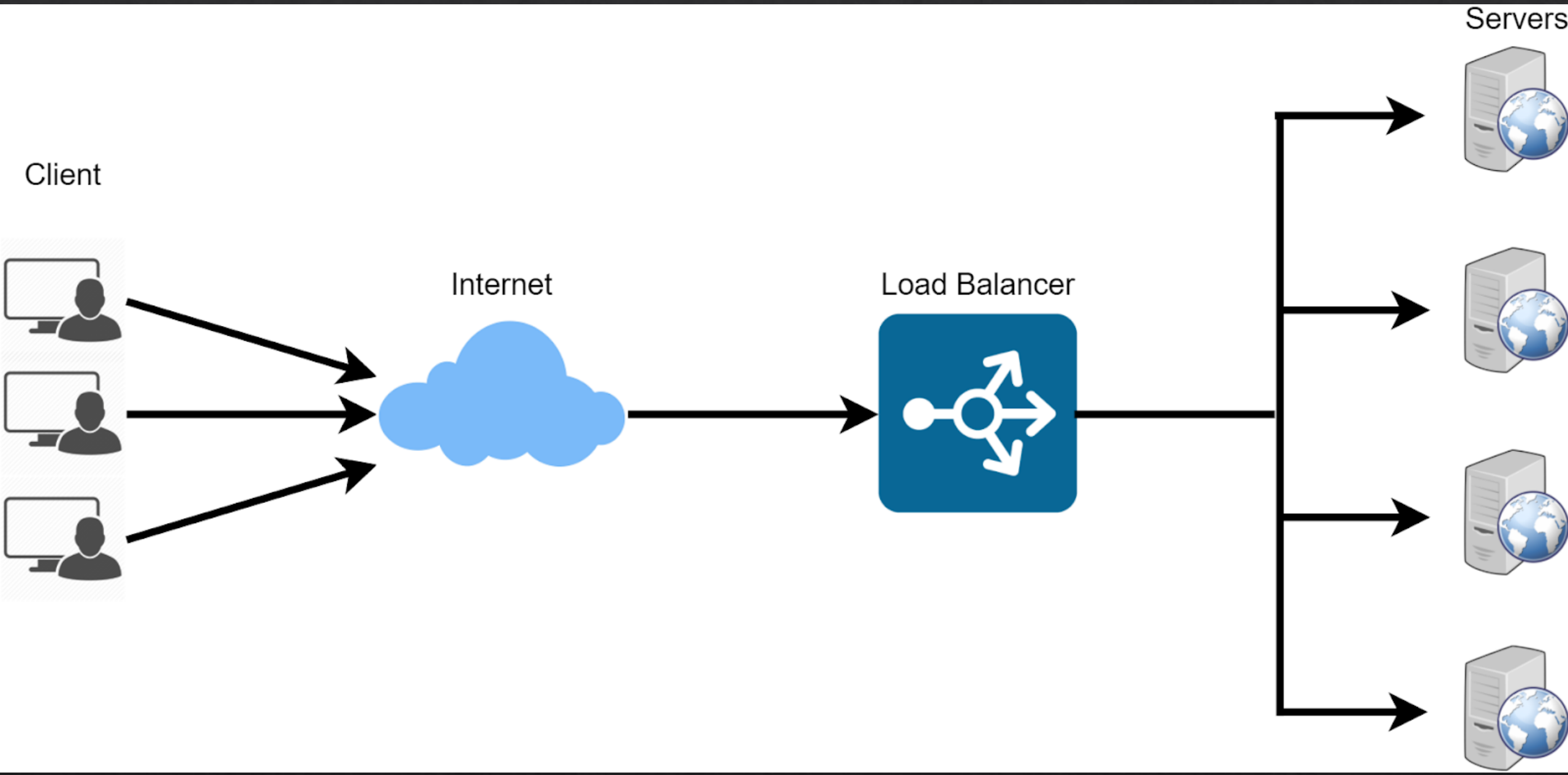
- If requests can be processed in parallel, the blocked time can be reduced.
- Concurrency can be introduced by processing different streams of events on different threads or by creating additional threads to process different sets of activities.
- Once concurrency has been introduced, you can choose scheduling policies to achieve the goals you find desirable using the schedule resources tactic.



Source: Stafman 2015

Maintain Multiple Copies of Computations

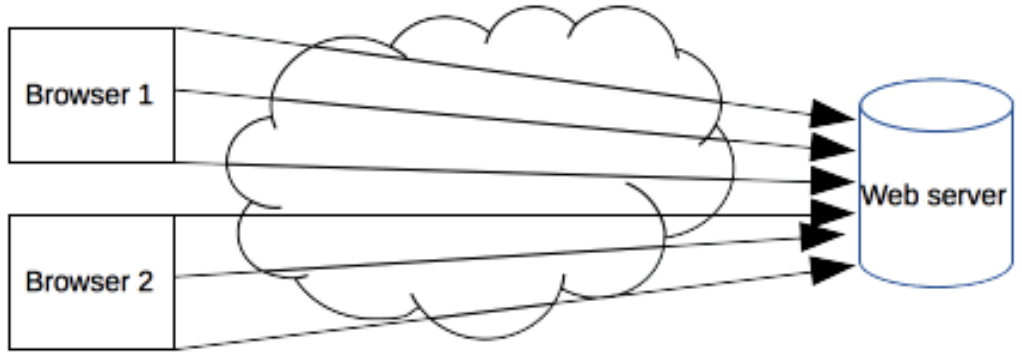
- This tactic reduces the contention that would occur if all requests for service were allocated to a single instance.
- Replicated services in a microservice architecture or replicated web servers in a server pool are examples of replicas of computation.
- A load balancer is a piece of software that assigns new work to one of the available duplicate servers; criteria for assignment vary but can be as simple as a round-robin scheme or assigning the next request to the least busy server.



Source: https://www.cloud4u.com/au-en/blog/what-is-a-load-balancer-and-its-types/

Maintain Multiple Copies of Data

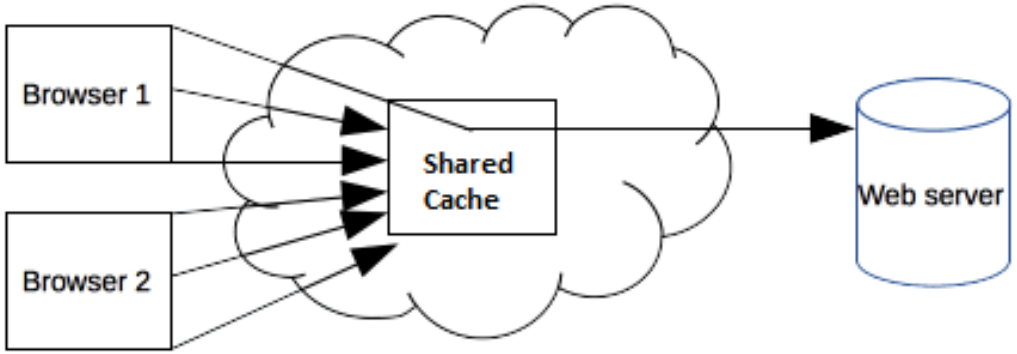
- Two common examples of maintaining multiple copies of data are data replication and caching.
- Data replication involves keeping separate copies of the data to reduce the contention from multiple simultaneous accesses. Because the data being replicated is usually a copy of existing data, keeping the copies consistent and synchronized becomes a responsibility that the system must assume.
- Caching also involves keeping copies of data (with one set of data possibly being a subset of the other), but on storage with different access speeds. The different access speeds may be due to memory speed versus secondary storage speed, or the speed of local versus remote communication. Another responsibility with caching is choosing the data to be cached.



All identical requests are
going through to the server.

Copy หมื่นครั้งที่เดียวกัน

Shared cache

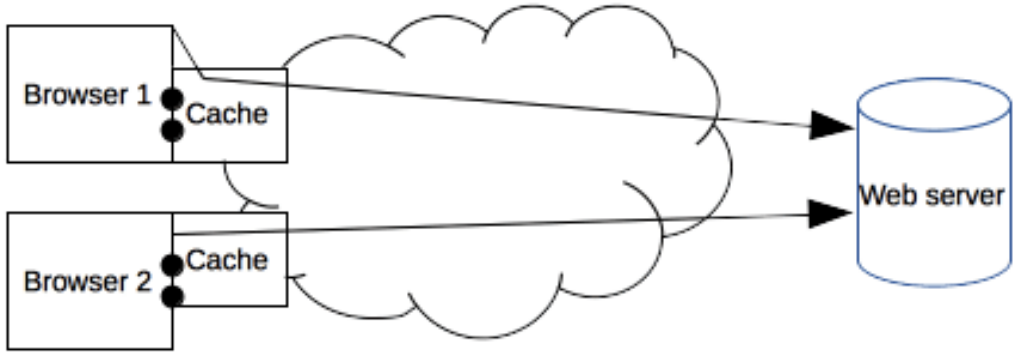


The first request is going through.

Subsequent identical requests are served
by the shared cache.

Copy ของใครของมัน

Local (private) cache



The first request of each client is going through.

Subsequent identical requests are not even sent, but
served by the local cache.

Source: https://whatdoiknow.org/website-caching/

Bound Queue Sizes

- This tactic controls the maximum number of queued arrivals and consequently the resources used to process the arrivals.
- If you adopt this tactic, you need to establish a policy for what happens when the queues overflow and decide if not responding to lost events is acceptable.
- This tactic is frequently paired with the limit event response tactic

กำหนด ความสำคัญ (Priority)

Schedule Resources

- A scheduling policy conceptually has two parts: a priority assignment and dispatching.
- All scheduling policies assign priorities.
- A high-priority event stream can be dispatched, assigned to a resource, only if that resource is available. Sometimes this depends on preempting the current user of the resource.
- Possible preemption options are as follows:
  - Can occur anytime
  - Can occur only at specific preemption points
  - Executing processes cannot be preempted

Schedule Resources: Scheduling Policies

Some common scheduling policies are:

- **First-in/first-out** queues treat all requests for resources as equals and satisfy them in turn.
- **Fixed-priority scheduling** assigns each source of resource requests a particular priority.
- **Dynamic priority scheduling**. Strategies include:
  - Round-robin orders the requests and assigns the resource to the next request in that order.
  - Earliest-deadline-first assigns priorities based on the pending requests with the earliest deadline.
  - Least-slack-first assigns the highest priority to the job having the least "slack time," which is the difference between the execution time remaining and the time to the job's deadline.
- **Static scheduling** is a scheduling strategy in which the preemption points and the sequence of assignment to the resource are determined offline.

SECURITY

Security

สถาวะที่จะพบทำงานได้ แม้ว่า ความเสี่ยงเข้ามา

- Security is a condition that results from the establishment and maintenance of protective measures that enable an organization to perform its mission or critical functions despite risks posed by threats to its use of systems.
- Protective measures may involve a combination of deterrence, avoidance, prevention, detection, recovery, and correction that should form part of the organization's risk management approach.

https://csrc.nist.gov/glossary/term/security

Security General Scenario (1/3)

Source of stimulus	Human or another system which may have been previously identified (either correctly or incorrectly) or may be currently unknown. A human attacker may be from outside the organization or from inside the organization.
Stimulus	Unauthorized attempt is made to display data, change or delete data, access system services, change the system's behavior, or reduce availability.
Artifacts	System services, data within the system, a component or resources of the system, data produced or consumed by the system
Environment	The system is either online or offline; either connected to or disconnected from a network; either behind a firewall or open to a network; fully operational, partially operational, or not operational.

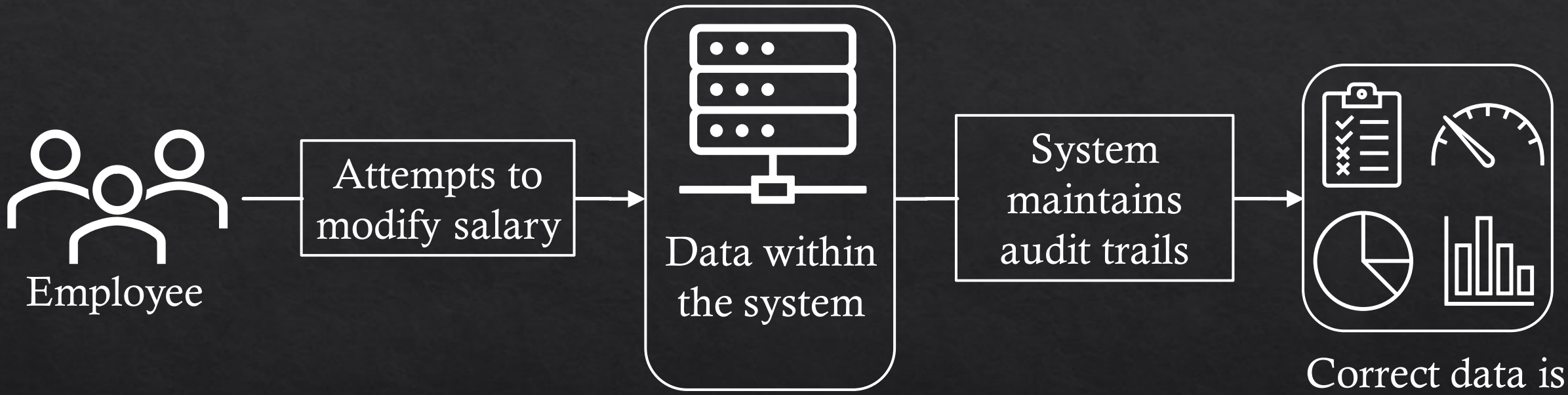
Security General Scenario (2/3)

Response	Transactions are carried out in a fashion such that • Data or services are protected from unauthorized access. • Data or services are not being manipulated without authorization. • Parties to a transaction are identified with assurance. • The parties to the transaction cannot repudiate their involvements. • The data, resources, and system services will be available for legitimate use. The system tracks activities within it by • Recording access or modification • Recording attempts to access data, resources, or services • Notifying appropriate entities (people or systems) when an apparent attack is occurring

Security General Scenario (3/3)

Response measure	One or more of the following: • How much of a system is compromised when a particular component or data value is compromised • How much time passed before an attack was detected • How many attacks were resisted • How long does it take to recover from a successful attack • How much data is vulnerable to a particular attack

Sample Security Scenario



Tactics for Security

Detect Attacks	Resist Attacks	React to Attacks	Recover from Attacks
• Detect intrusion • Detect service denial • Verify message integrity • Detect message delivery anomalies	• Identity actors • Authenticate actors • Authorize actors • Limit access • Limit exposure • Encrypt data • Separate entities • Validate input • Change credential settings	• Revoke access • Restrict login • Inform actors	• Audit • Nonrepudiation

More on Security, but…

- We will not cover more of security here in our course.
- Security is a very broad topic with many subtopics.
- It is worth its own courses!!

ความสามารถในการทดสอบ

TESTABILITY

Testability

- Software testability refers to the ease with which software can be made to demonstrate its faults through (typically execution-based) testing.
- Specifically, testability refers to the probability, assuming that the software has at least one fault, that it will fail on its next test execution.
- Intuitively, a system is testable if it "gives up" its faults easily. If a fault is present in a system, then we want it to fail during testing as quickly as possible.

Testability General Scenario (1/3)

Source of stimulus	• Unit testers • Integration testers • System testers • Acceptance testers • End users Either running tests manually or using automated testing tools
Stimulus	A test or set of tests is initiated. These tests serve to: • Validate system functions • Validate qualities • Discover emerging threats

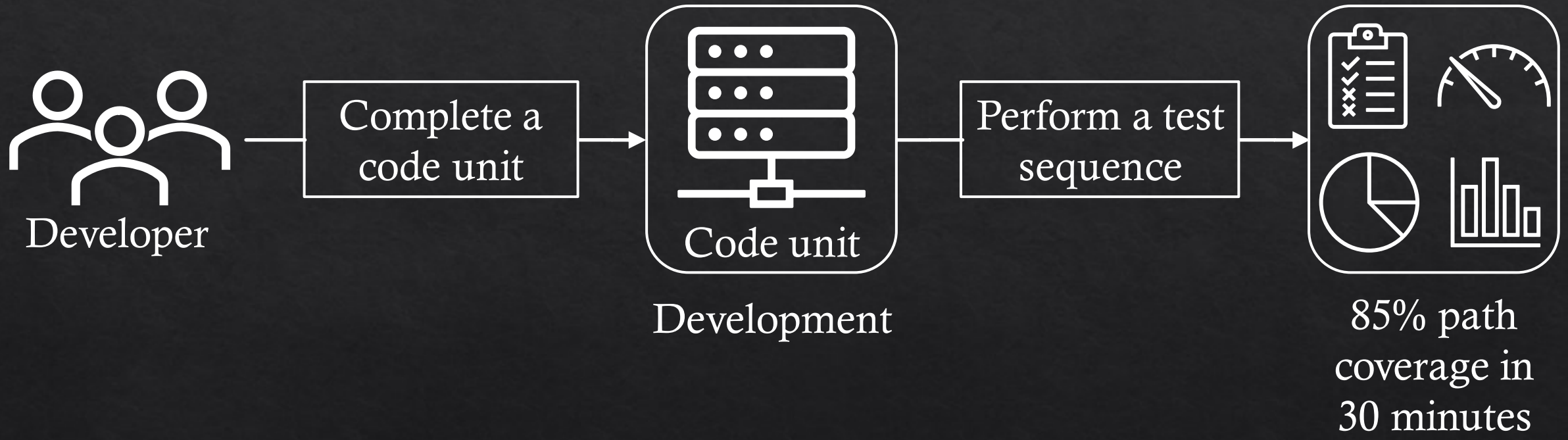
Testability General Scenario (2/3)

Artifacts ระบบที่จะเทสได้	The portion being tested: • A unit of code (corresponding to a module in the architecture) • Components • Services • Subsystems • The entire system The test infrastructure
Environment ตัวอย่างที่เทส ที่เกิดขึ้น	The set of tests is executed due to: • The completion of a coding increment such as a class, layer, or service • The completed integration of a subsystem • The complete implementation of the whole system • The deployment of the system into a production environment • The delivery of the system to a customer • A testing schedule

Testability General Scenario (3/3)

Response	One or more of the following: • Execute test suite and capture results • Capture activity that resulted in the fault • Control and monitor the state of the system
Response measure	One or more of the following: • Effort to find a fault or class of faults • Effort to achieve a given percentage of state space coverage • Probability of fault being revealed by the next test • Time to perform tests • Effort to detect faults • Length of time to prepare test infrastructure • Effort required to bring the system into specific state • Reduction in risk exposure (size(loss) × prob(loss))

Sample Testability Scenario



Tactics for Testability

Control and Observe System State	Limit Complexity
• Specialized interfaces • Record/Playback • Localize state storage • Abstract data sources มอดอัพ ถอดท้ายรถ • Sandbox • Executable assertions	• Limit structural complexity เชิงโครงสร้างไม่ซับซ้อน • Limit nondeterminism ต้องอยู่ใน outcome ตามที่กำหนด

More of Testability, but…

- The topic consist of many subtopics.
- It is worth its own courses!!

ก. สะดวกในการใช้งาน

USABILITY

Usability

- Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides. Usability comprises the following areas:
  - **Learning System Features**: What can the system do to make the task of learning easier?
  - **Using a System Efficiently**: What can the system do to make the user more efficient in its operation?
  - **Minimizing the Impact of User Errors**: What can the system do to ensure that a user error has minimal impact?
  - **Adapting the System to User Needs**: How can the user (or the system itself) adapt to make the user's task easier?
  - **Increasing Confidence and Satisfaction**: What does the system do to give the user confidence that the correct action is being taken?

Usability General Scenario (1/3)

Source of stimulus	The end user (who may be in a specialized role), such as a system or network administrator) is the primary source of the stimulus for usability. An external event arriving at a system (to which the user may react) may also be a stimulus source.
Stimulus	End user wants to: • Learn to use the system • Use a system efficiently • Minimize the impact of errors • Adapt the system • Configure the system.

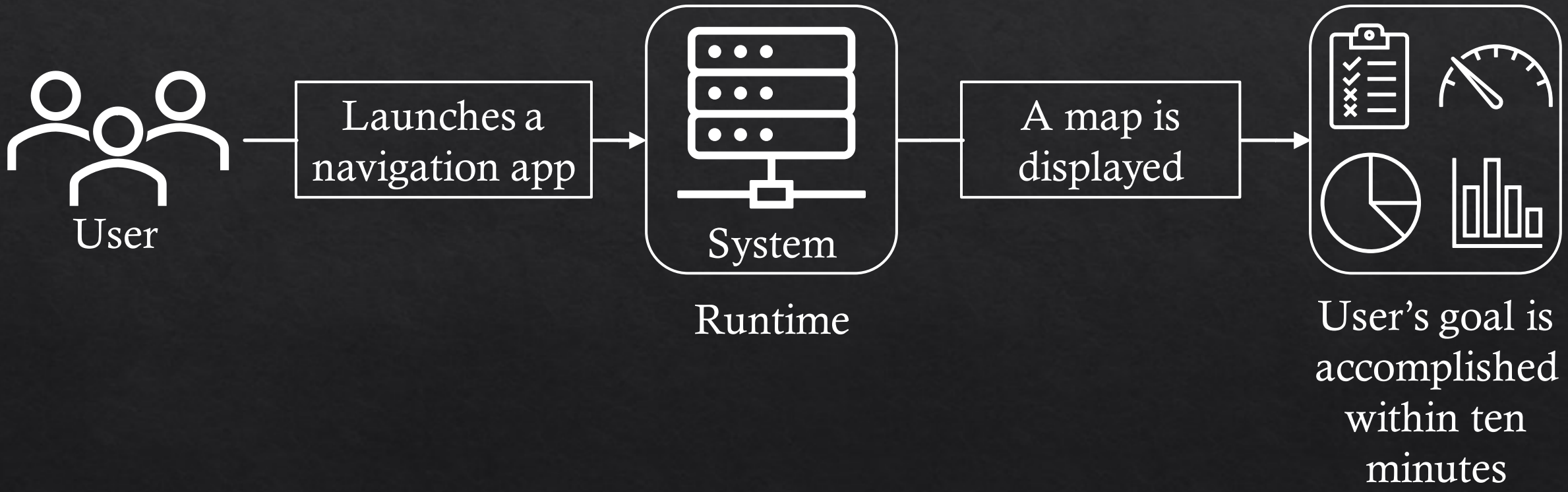
Usability General Scenario (2/3)

Artifacts	Common examples include: • A GUI • A command-line interface • A voice interface • A touch screen
Environment	The user actions with which usability is concerned always occur at runtime or system configuration time.

Usability General Scenario (3/3)

Response	The system should: • Provide the user with the features needed User ได้ทุกสิ่งที่ต้องการ • Anticipate the user's needs • Provide appropriate feedback to the user
Response measure	One or more of the following: • Task time • Number of errors • Number of tasks accomplished • User satisfaction • Gain of user knowledge • Ratio of successful operations to total operations สัดส่วนที่ทำสำเร็จ • Amount of time or data lost when an error occurs

Sample Usability Scenario



Tactics for Usability

Support User Initiative	Support System Initiative
• Cancel • Undo • Pause/Resume • Aggregate	• Maintain task model • Maintain user model • Maintain system model

Support User Initiative

- **Cancel**: The activity being canceled must be terminated; any resources being used by the canceled activity must be freed.
- **Undo**: The system must maintain a sufficient amount of information about system state so that an earlier state may be restored.
- **Pause/Resume**: Pausing a long-running operation may be done to temporarily free resources so that they may be reallocated to other tasks.
- **Aggregate**: When a user is performing repetitive operations, or operations that affect a large number of objects in the same way, it is useful to provide the ability to aggregate the lower-level objects into a single group, so that the operation may be applied to the group.

ระบบเริ่ม

Support System Initiative (1/2)

- When the system takes the initiative, it must rely on a model of the user, the task being undertaken by the user, or the system state itself.
- Each model requires various types of input to accomplish its initiative. เพื่อตอบสนอง User
- The support system initiative tactics are those that identify the models the system uses to predict either its own behavior or the user's intention.

ม ว ส

Support System Initiative (2/2)

- **Maintain Task Model**: The task model is used to determine context so the system can have some idea of what the user is attempting to do and provide assistance. For example, many search engines provide predictive type-ahead capabilities, and many mail clients provide spell-correction.
- **Maintain User Model**: This model explicitly represents the user's knowledge of the system, the user's behavior in terms of expected response time, and other aspects specific to a user or a class of users. For example, language-learning apps are constantly monitoring areas where a user makes mistakes and then providing additional exercises to correct those behaviors.
- **Maintain System Model**: The system maintains an explicit model of itself. This is used to determine expected system behavior so that appropriate feedback can be given to the user. A common manifestation of a system model is a progress bar that predicts the time needed to complete the current activity.

More on Usability, but…

- You better learn more from courses on HCI (Human-Computer Interaction), UX (user experience) and UI (user interface).

Summary

- Quality Attribute Scenario
- Availability
- Integrability
- Modifiability
- Performance
- Security (briefly)
- Testability (briefly)
- Usability (briefly)

Modifiability

Performance

กล่าวๆ

Recommended Resources

- Software Architecture in Practice: Distinguish Functionality from Quality Attributes – YouTube