51.505 – Foundations of Cybersecurity

Week 2 – Security Policies

Created by **Martin Ochoa** (2017)
Modified by **Jianying Zhou** (2018)

Last updated: 13 Sept 2018

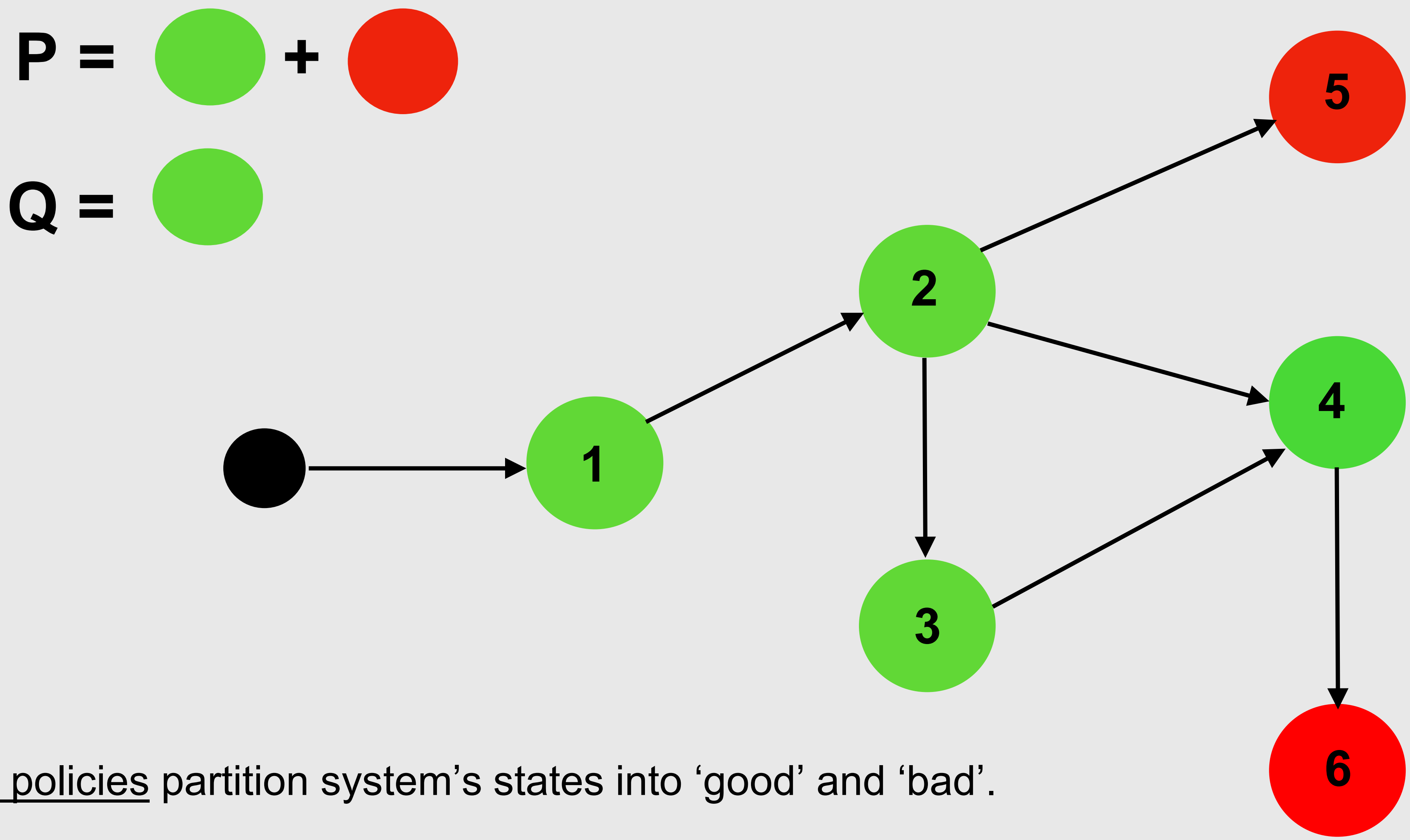
Recap

- Questions on last week’s exercises?

Security Policy & Mechanism

- *A* <u>security policy</u> *is a statement of what is, and is not, allowed.*
  - ✓ Security policies can be thought of as requirements to a system, or refinements of more abstract properties.
- *A* <u>security mechanism</u> *is a method, tool or procedure for enforcing a security policy.*
  - ✓ Mechanisms are ways to enforce policies. Broadly 3 classes:
    - ❖ Prevention
    - ❖ Detection
    - ❖ Recovery

Policies & System's States



- Security policies partition system's states into 'good' and 'bad'.
- Security mechanisms enforce systems within 'good' states.
- A secure system is a system that starts in a 'good' (authorized) state and cannot enter a 'bad' (unauthorized) state.
- Is this a secure system?

Example

- **University policy disallows cheating,**
  - ✓ Includes copying homework, with or without permission.
- CS class has students do homework on computer.
  - ✓ Anne forgets to read-protect her homework file.
  - ✓ <u>Bill copies it.</u>
- Who cheated?
  - ✓ Anne, Bill, or both?

Types of Access Control

- *Discretionary Access Control (DAC):* individual user sets access control mechanism to allow or deny access to an object.
- *Mandatory Access Control (MAC):* system mechanism controls access to object, and individual cannot alter that access.

Policy Languages

- Express security policies in a precise way.
- *High-level languages*
  - ✓ Policy constraints expressed abstractly.
- *Low-level languages*
  - ✓ Policy constraints expressed in terms of program options, input, or specific characteristics of entities on system.

High-Level Policy Language

- Access constraints: **deny**(*s op x*) when *b*
  - ✓ *s = subject*
  - ✓ *x = subject or class*
  - ✓ *b = Boolean expression*
  - ✓ *op* = -| (create an instance) *or* |→ (execute an object)

➔ ***s*** cannot perform operation ***op*** on ***x*** when condition ***b*** is true.

- Ignores implementation issues.

Example

- Downloaded program cannot access password file on UNIX system.
- Program's class and methods for files:

```
class File {
   public file(String name);
   public String getfilename();
   public char read();
```

- Constraint:

```
deny( |-> file.read) when
   (file.getfilename() == "/etc/passwd")
```

Confidentiality

- *"the **property**, that information is not made available or disclosed to unauthorized individuals, entities, or processes"* (ISO27000).
- Formally: X = set of entities, I = information
- I has confidentiality property with respect to X if no $x \in X$ can obtain information from I.
- Example:

  ✓ X = set of students.

  ✓ I = final exam answers.

  ✓ I is confidential with respect to X if students cannot obtain final exam answer keys.

Security & Precision

- *Program*: a function with multiple inputs and one output

  Let p be a function $\boxed{p: I_1 \times \ldots \times I_n \rightarrow R}$. Then p is a program with n inputs $i_k \in I_k$, $1 \leq k \leq n$, and one output $r \rightarrow R$

- Example: $sum(i,j,k) = i+j+k$

Protection Mechanism

- A *protection mechanism* is defined formally as

Let p be a function $p: I_1 \times \ldots \times I_n \rightarrow R$. A protection mechanism m is a function

$$\boxed{m: I_1 \times \ldots \times I_n \rightarrow R \cup E}$$

for which, when $i_k \in I_k$, $1 \leq k \leq n$, either

– $m(i_1, \ldots, i_n) = p(i_1, \ldots, i_n)$ or

– $m(i_1, \ldots, i_n) \in E$. **(Error)**

Confidentiality Policy

- A *confidentiality policy* prevents unauthorized disclosure of information.

Formally, for $\boxed{p: I_1 \times \ldots \times I_n \rightarrow R}$, it is a function $c: I_1 \times \ldots \times I_n \rightarrow A$, where $A \subseteq I_1 \times \ldots \times I_n$
A is set of inputs available to observer

For instance, $c(i,j,k) = (i,j)$: i, j can be disclosed but k is kept confidential.

- A *security mechanism* (***m***) conforms to a stated *confidentiality policy* (***c***):

$$m \text{ secure iff } \exists\, m' : A \rightarrow R \cup E \text{ such that, for all } i_k \in I_k,\ 1 \leq k \leq n:$$
$$m(i_1, \ldots, i_n) = m'(c(i_1, \ldots, i_n))$$

Examples

- $c(i_1, ..., i_n) = C$, a constant ➔ $m = C$
  - ✓ Deny observer any information. (Output does not vary with inputs.)
- $c(i_1, ..., i_n) = (i_1, ..., i_n)$, and $m' = m = p$
  - ✓ Allow observer full access to information.
- $c(i_1, ..., i_n) = i_1, i_2$
  - ✓ Allow observer information about first two inputs but no information about other inputs.

Precision

- A security mechanism can be overly restrictive.

m_1 as precise as m_2 $\boxed{(m_1 \approx m_2)}$ if, for all inputs $i_1, \ldots, i_n$,

$$m_2(i_1, \ldots, i_n) = p(i_1, \ldots, i_n) \Rightarrow m_1(i_1, \ldots, i_n) = p(i_1, \ldots, i_n)$$

m_1 more precise than m_2 $\boxed{(m_1 \sim m_2)}$ if there is an input $(i_1', \ldots, i_n')$ such that $m_1(i_1', \ldots, i_n') = p(i_1', \ldots, i_n')$ and $m_2(i_1', \ldots, i_n') \neq p(i_1', \ldots, i_n')$.

- Example: Suppose $c(i_1, i_2, i_3) = i_1, i_2,$
  - ✓ $m_1(i,j,k)=i+j$ is more precise than $m_2(i,j,k) = i$.
  - ✓ Why ?

Key Points

- Policies describe *what* is allowed.
- Mechanisms control *how* policies are enforced.
- Trust underlies everything.

Bell-LaPadula Model

- Goal of confidential policy: prevent the unauthorized disclosure of information.
  - ✓ Deal with information flow
  - ✓ Integrity incidental
- Bell-LaPadula Model
  - ✓ Military style classification
  - ✓ Significant influence in computer security

Step 1: Security Level

- Security levels are arranged in linear ordering.
  - ✓ Top Secret: highest
  - ✓ Secret
  - ✓ Confidential
  - ✓ Unclassified: lowest
- Subjects (*s*) have security clearance *L(s).*
- Objects (*o*) have security classification *L(o).*

Example

Security Level	Subject	Object
Top Secret (TS)	Tamara, Thomas	Personnel Files
Secret (S)	Sally, Samuel	Email files
Confidential (C)	Claire, Clarence	Activity Logs
Unclassified (UC)	Ulaley, Ursula	Telephone Lists

- Tamara can read all files.
- Claire cannot read Personnel or E-Mail Files.
- Ulaley can only read Telephone Lists.

Reading Information

- Information flows up, not down
  - ✓ "Reads up" disallowed, "reads down" allowed.
- Simple Security Condition (Step 1)
  - ✓ Subject *s* can **read** object *o* iff, $L(o) \le L(s)$ and *s* has permission to read *o.*
  - ✓ Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission).
  - ✓ Sometimes called "no reads up" rule.
- Example: What files Samuel can read?

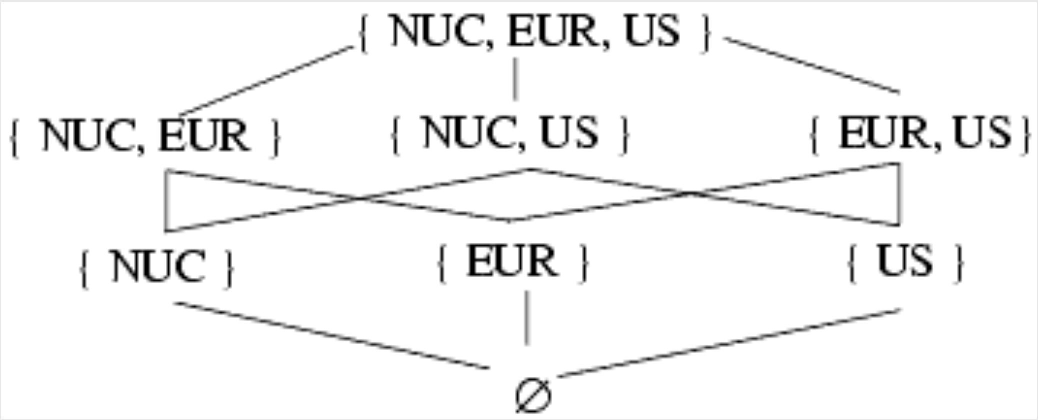
Writing Information

- Information flows up, not down
  - ✓ "Writes up" allowed, "writes down" disallowed.
- *-Property (Step 1)
  - ✓ Subject *s* can **write** object *o iff* $L(s) \leq L(o)$ and *s* has permission to write *o.*
  - ✓ Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission).
  - ✓ Sometimes called "no writes down" rule.
- Example: What files Samuel can write?

Step 2: Extension to Categories

- Expand notion of security level to include <u>categories</u>
- Security level is (clearance, category set)
- Examples:
  - ✓ (Top Secret, { NUC, EUR, ASI })
  - ✓ (Confidential, { EUR, ASI })
  - ✓ (Secret, { NUC, ASI })

Levels & Lattices



- (L, C) **dom** (L', C') iff L' ≤ L and C' ⊆ C
- Examples:
  - ✓ (Top Secret, {NUC, ASI}) dom (Secret, {NUC})
  - ✓ (Secret, {NUC, EUR}) dom (Confidential,{NUC, EUR})
  - ✓ (Top Secret, {NUC}) ¬ dom (Confidential, {EUR})

Reading Information

- Information flows up, not down
  - ✓ “Reads up” disallowed, “reads down” allowed.
- Simple Security Condition (Step 2)
  - ✓ Subject *s* can **read** object *o* iff *L(s)* dom *L(o)* and *s* has permission to read *o.*
  - ✓ Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission).
  - ✓ Sometimes called “no reads up” rule.

Writing Information

- Information flows up, not down
  - ✓ "Writes up" allowed, "writes down" disallowed.
- *-Property (Step 2)
  - ✓ Subject s can **write** object *o* iff *L(o) dom L(s)* and *s* has permission to write *o.*
  - ✓ Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission).
  - ✓ Sometimes called "no writes down" rule.

Example

- George is cleared into security level (Secret, { NUC, EUR}).
- Doc_A is classified as (Confidential, { NUC }).
- Doc_B is classified as (Secret, { EUR, US}).
- Doc_C is classified as (Top Secret, { NUC, EUR }).

- What documents does George have read access?
- What documents does George have write access?

Integrity Policy

Inspired by the following commercial requirements:

1. Users will not write their own programs, but will use existing production programs and databases.
2. Programmers will develop and test programs on a non-production system; if they need access to actual data, they will be given production data via a special process, but will use it on their development system.
3. A special process must be followed to install a program from the development system onto the production system.
4. The special process in requirement 3 must be controlled and audited.
5. The managers and auditors must have access to both the system state and the system logs that are generated.

Biba Integrity Model

- A system consists of a set *S* of subjects, a set *O* of objects, and a set *I* of integrity levels.
- The relation $\leq \subseteq I \times I$ holds when the second integrity level *dominates* the first.
- $min: I \times I \rightarrow I$ gives the lesser of two integrity levels (with respect to $\leq$).
- $i: S \cup O \rightarrow I$ returns the integrity level of an object or a subject.
- $r \subseteq S \times O$ defines the ability of a subject to <u>read</u> an object.
- $w \subseteq S \times O$ defines the ability of a subject to <u>write</u> to an object.
- $x \subseteq S \times S$ defines the ability of a subject to invoke (<u>execute</u>) another subject.

Integrity Levels

- The higher the integrity level, the more confidence.
  - ✓ That a program will execute correctly.
  - ✓ That data is accurate and/or reliable.
- Note relationship between *integrity* and *trustworthiness.*
- Important point: integrity levels are not security levels. Example:
  - ✓ Highly trusted
  - ✓ Medium trust
  - ✓ Low trust

Low-Water-Mark Policy

- Rules

  1. $s \in S$ can <u>write</u> to $o \in O$ if and only if $i(o) \leq i(s)$.

  2. If $s \in S$ <u>reads</u> $o \in O$, then $i'(s) = min(i(s), i(o))$,
     where $i'(s)$ is the subject's integrity level after the read.

  3. $s_1 \in S$ can <u>execute</u> $s_2 \in S$ if and only if $i(s_2) \leq i(s_1)$.

- What is intuition behind these rules?

Problems

- Subjects’ integrity levels decrease as system runs.
  - ✓ Soon no subject will be able to access objects at high integrity levels.
- Crux of problem is the model prevents *indirect modification*.
  - ✓ Because subject levels lowered when subject reads from low-integrity object.

Ring Policy

- Idea: keep subject levels static.
- Rules
    1. $s \in S$ can <u>write</u> to $o \in O$ if and only if $i(o) \leq i(s)$.
    2. Any subject can read any object.
    3. $s_1 \in S$ can <u>execute</u> $s_2 \in S$ if and only if $i(s_2) \leq i(s_1)$.
- Eliminate the indirect modification problem.
- Problem?

Strict Integrity Model

- Similar to Bell-LaPadula model
- Rules

  1. $s \in S$ can <u>read</u> $o \in O$ if and only if $i(s) \le i(o)$. → "<u>no reads down</u>"
  2. $s \in S$ can <u>write</u> to $o \in O$ if and only if $i(o) \le i(s)$. → "<u>no writes up</u>"
  3. $s_1 \in S$ can <u>execute</u> $s_2 \in S$ if and only if $i(s_2) \le i(s_1)$.

- Information flow result holds, but its proof changes.
- Term "Biba's Model" refers to this.

Lipner's Model

- Lipner proposed this as first realistic commercial model.
    - ✓ Combine Bell-LaPadula and Biba models to obtain a model conforming to the commercial requirements.
- Do it in two steps:
    - ✓ Bell-LaPadula component first (control reading)
    - ✓ Add in Biba component (control writing)

Bell-LaPadula Security

- 2 security clearances (higher to lower):
  - ✓ AM (Audit Manager): system audit, management functions
  - ✓ SL (System Low): any process can read at this level
- 5 categories:
  - ✓ D (Development): production programs in development but not yet in use
  - ✓ PC (Production Code): production processes, programs
  - ✓ PD (Production Data): data covered by integrity policy
  - ✓ SD (System Development): system programs in development but not yet in use
  - ✓ T (Software Tools): programs on production system not related to protected data

Ideas

- <u>Ordinary users</u> can execute (read) production code but cannot alter it.
- <u>Ordinary users</u> can alter and read production data.
- <u>System managers</u> need access to all logs but cannot change security levels of objects.
- <u>System controllers</u> need to install code (hence downgrade capability).
- <u>Logs</u> are append only, so must dominate subjects writing them.

Subjects & Security Levels

Subject	Security Level
Ordinary users	(SL, { PC, PD })
Application developers	(SL, { D, T })
System programmers	(SL, { SD, T })
System managers and auditors	(AM, { D, PC, PD, SD, T })
System controllers	(SL, { D, PC, PD, SD, T }) and downgrade privilege

Objects & Security Levels

Object	**Security Level**
Development code / test data	(SL, { D, T })
Production code	(SL, { PC })
Production data	(SL, { PC, PD })
Software tools	(SL, { T })
System programs	(SL, Ø)
System programs in modification	(SL, { SD, T })
System and application logs	(AM, { appropriate categories })

Meet Requirements?

1. Users have no access to T, so cannot write their own programs.

2. System programmers have no access to PD, so cannot access production data; If needed, it must be put into D (downgrade), requiring the system controller to intervene.

3. Installing a program requires downgrade procedure (from D to PC), so only system controllers can do it.

4. Control: only system controllers can downgrade; audit: any such downgrading must be logged.

5. System management and audit users are in AM and so have access to system state and logs.

Problems

- Too inflexible.
  - ✓ System managers cannot run programs for repairing inconsistent or erroneous production database.
  - ✓ System managers at AM, production data at SL. ("No writes down" rule)
- So add more …

Adding Integrity

- 3 integrity classifications (highest to lowest):
  - ✓ ISP (System Program): for system programs
  - ✓ IO (Operational): production programs, development software
  - ✓ ISL (System Low): users get this on log in
- 2 integrity categories:
  - ✓ ID (Development): development entities
  - ✓ IP (Production): production entities

Simplified Bell-LaPadula

- Reduce security categories to 3:
  - ✓ SP (Production): production code, data
  - ✓ SD (Development): same as old D
  - ✓ SSD (System Development): same as old SD

Subjects & Levels

Subject	Security Level (simplified) <control reading>	Integrity Level <control writing>
Ordinary users	(SL, { SP })	(ISL, { IP })
Application developers	(SL, { SD })	(ISL, { ID })
System programmers	(SL, { SSD })	(ISL, { ID })
System managers and auditors	(AM, { SP, SD, SD, SSD })	(ISL, { IP, ID })
System controllers	(SL, { SP, SD, SSD }) and downgrade privilege	(ISP, { IP, ID })
Repair	(SL, { SP })	(ISL, { IP })

Objects & Levels

Object	**Security Level (simplified)** **<control reading>**	**Integrity Level** **<control writing>**
Development code / test data	(SL, { SD })	(ISL, { IP })
Production code	(SL, { SP })	(IO, { IP })
Production data	(SL, { SP })	(ISL, { IP })
Software tools	(SL, Ø)	(IO, { ID })
System programs	(SL, Ø)	(ISP, { IP, ID })
System programs in modification	(SL, { SSD })	(ISL, { ID })
System and application logs	(AM, { appropriate categories })	(ISL, Ø)
Repair	(SL, { SP })	(ISL, { IP })

Result

- Security clearances of subjects same as without integrity levels.
- Ordinary users need to modify production data, so ordinary users must have write access to integrity category IP.
- Ordinary users must be able to write production data but not production code; integrity classes allow this.
- Note: writing constraints removed from security classes.

Key Points

- Security labels limit the flow of information.
- Integrity labels inhibit the modification of information.
- Bell-LaPadula Model for *confidential policies*.
  - ✓ 1st mathematical model
  - ✓ Military style classification
- Biba Model for *integrity policies*.
- Lipner's Model
  - ✓ 1st realistic commercial model
  - ✓ Combine Bell-LaPadula Model and Biba Model to conform to the commercial requirements.

Exercises & Reading

- Classwork (Exercise Sheet 2): due on Fri Sept 21, 10:00 PM
- Homework (Exercise Sheet 2): due on Fri Sept 28, 6:59 PM
- Reading: MB [Ch4 (without Theorem 4-3), Ch5 (without 5.2.3, 5.2.4, 5.3, 5.4), Ch6 (without 6.4)]

End of Slides for Week 2