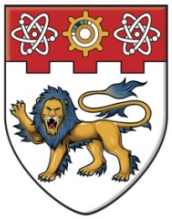


NANYANG TECHNOLOGICAL UNIVERSITY

Modeling Software Structure using UML

presented by

Shar Lwin Khin
Research Scientist
SCSE
lkshar@ntu.edu.sg
N4-02c-76

Courtesy of Kevin Anthony Jones' slides

Foreword

Objective of this lesson is to develop student's proficiency in *modeling architecture* using UML diagrams

UML is defacto standard for designing, communicating, and documenting software being developed. UML has separate models for structure & behavior (arch think concepts). There are two diagrams for modeling architecture: Component for structure, and Communication for behavior

This lesson leverages course 2006 on the background of UML in software development, and the preceding lesson as explaining all requisite details about software components

In this lesson, we will study only the Component diagram

This lesson ends with a mini-workshop; let's begin

How is this relevant to me?

Usage of UML in software development

Relative popularity of UML diagrams

NANYANG TECHNOLOGICAL UNIVERSITY

High usage of UML in sw development

The results of a developer survey carried out by Computerworld and subsequently published in their Mar 28, 2005 edition, 33% reported that UML is in use at their organization and another 13% said that there plans for future use

A survey conducted by BZ Research in late Jun 2005 indicated that 34% of developers presently use UML-based modeling for applications development; of that group, 2% said they use UML to model all applications, and 32% said that UML is used only for modeling some applications

In Oct 2010 Paulo Merson from SEI posted, "UML is a well-known visual language that can capture much of the information that one needs to communicate about the architecture"

In March 2011, Systems Flow Inc posted, "Our team has experienced great success with visual modeling based on the Unified Modeling Language; this is authenticated *Leveraging UML as a Standard Notation for Enterprise Architecture*"

32

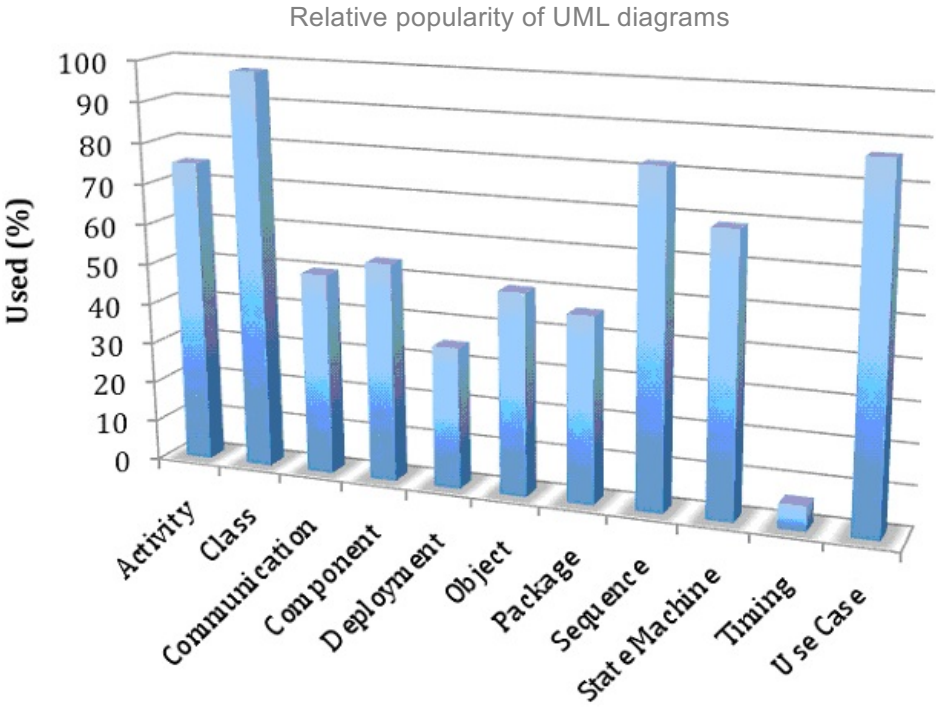
33

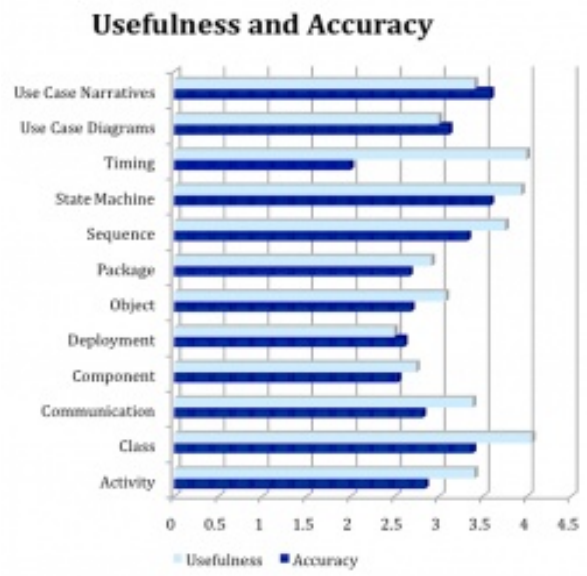
34

A recent survey conducted by Dwayne Anius and Brian Dobing on the topic "Programmer's views on the usefulness of UML diagrams" showed the use of the different UML diagrams

Grady Booch posted in Apr 2006, "if we consider the worldwide developer market according to IDC is approximately 13 million or so IT professionals, that's a great penetration. UML is indeed the open standard of choice for modeling"

Scott W. Ambler of Ambysoft Inc wrote, "Component diagrams are particularly useful with larger teams. Once the interfaces are defined, and agreed to by your team, it makes it much easier to organize the development effort between subteams"



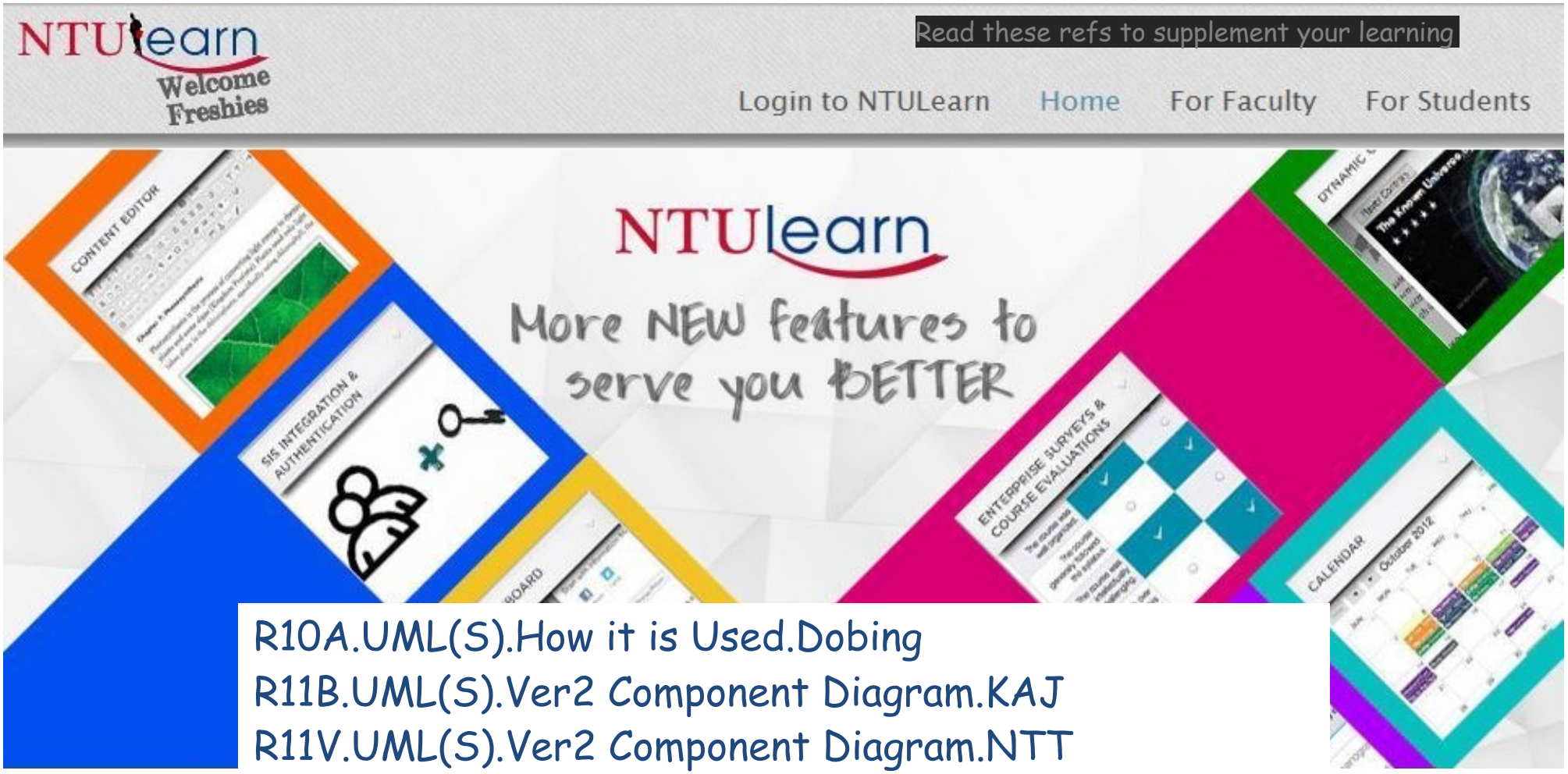


http://blog.architexa.com/2010/06/a-glimpse-into-the-current-state-of-uml-use/

NANYANG TECHNOLOGICAL UNIVERSITY

As a student of SSAD course, by end of this session, you will be able to …

… model according to proper notation and semantics the 'structure' of a software architecture using UML Component diagram



R10A.UML(S).How it is Used.Dobing
R11B.UML(S).Ver2 Component Diagram.KAJ
R11V.UML(S).Ver2 Component Diagram.NTT

E41A.UML.In Practice.Petre
E42B.UML.Glimpse Into Current State of Its Use.Rosen
E43K.UML.Ver2 Spec Superstructure 04-07-05.OMG
E51V.UML.Ver2 Component Diagram.VTC

NANYANG TECHNOLOGICAL UNIVERSITY

Bloom's degree of learning

Final suggested learning consideration: expected depth of learning for the topic. Eminent educationalist Benjamin Bloom identified six degrees of cognitive assimilation of new knowledge: first is the lowest, most superficial degree, and sixth is the highest, deepest, most pervasive

The learning for this topic is *third* degree, *application* : student must correctly use notation and semantics of the UML Component diagram to model (draw + layout + annotate in friendly-readable form) architectures in a consistent manner for different scenarios

Assessing 'application': How to design or communicate or document architecture with…? What would result if… was used for [something else]? What examples can you find of… [doing something]? How would you organize… to [do something]?

Examinability: You are expected to *draw* from scratch, or *redraw* from an initial architecture, a UML Component diagram with correct notation and appropriate semantic at junior-level architecting

Prepare for incoming!

aka., activate your cognitive foundation for today's lesson

Vertex, edge, shape, symbol, and icon

Remind me, black and white box views

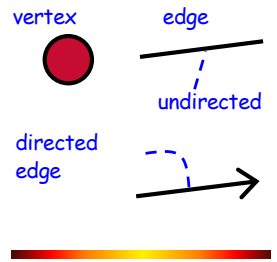
Remind me, unified modeling language (UML) (course 2006)

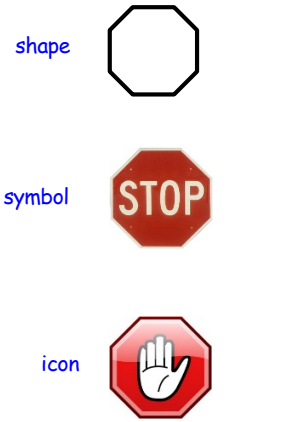
Class is key ModelElement in UML (course 2006)

Remind me about Interface in OOP (course 2002)

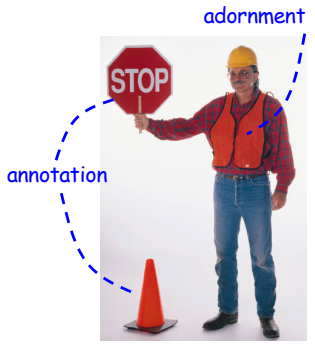
Vertex, edge, shape, symbol, and icon

Graph is a planar (2-D) arrangement of *vertices* connected by *edges* [see right]. Both vertices and edges may have contextual meaning and contain information. [see below] A vertex can either be a *shape*, *sym-bol*, or *icon*. An edge is a curve that is *directed* or *undirected*. Both vertices and edges may have distinct annotations and adornments





- Shape – 2-D outline enclosing an area
- Symbol – shape with text or text alone that represents/ means something without the necessity of having the likeness/appearance of the entity that it is representing
- Icon – picture-like shape that represents/means some-thing by having the likeness or being an analogy of the entity it is representing
- Adornment – semantic-less decoration of a target symbol used to enhance the recognition of the semantics of that symbol. It is often optional
- Annotation – shape or textual addition to a target symbol giving explanation/'semantic detail' to a target symbol. As such, the annotation has integral semantics and notation



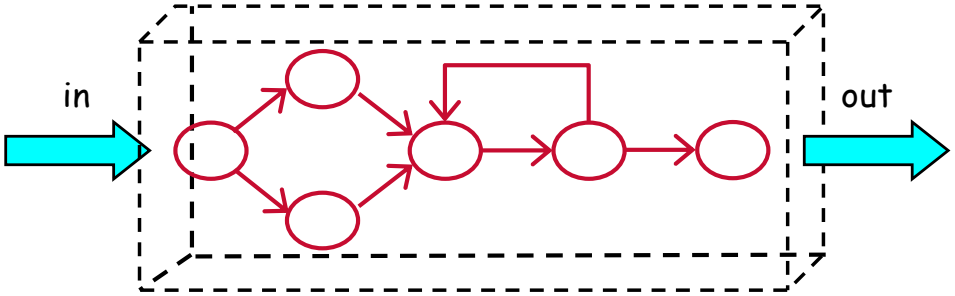
NANYANG TECHNOLOGICAL UNIVERSITY

Remind me, black and white box views

- Black-box – inner workings hidden; vertex perceived as closed machine consuming input and generating output
- White-box – inner workings exposed; vertex perceived as structure where its internal parts and their distinct behaviors are visible

Black-box

White-box



Eg., black and white-box views using vertices in previous slide …

NANYANG TECHNOLOGICAL UNIVERSITY

Remind me, unified modeling language (UML)

From course 2006 "Software Engineering", UML is a modelling language to design, communicate, and document software in development and fielding. It was created by Grady Booch, Ivor Jacobson, and Jim Rumbaugh at Rational Software Corporation. Version 0.8 was released in 1996, version 1 in 1997, version 2 in 2005, and the latest, version 2.5, in 2015

UML specifies 14 diagrams for software modelling, seven for structure, and seven for behaviour. These diagrams have been found to be exceptionally useful, and there are unlimited sample diagrams on the web to learn from and use

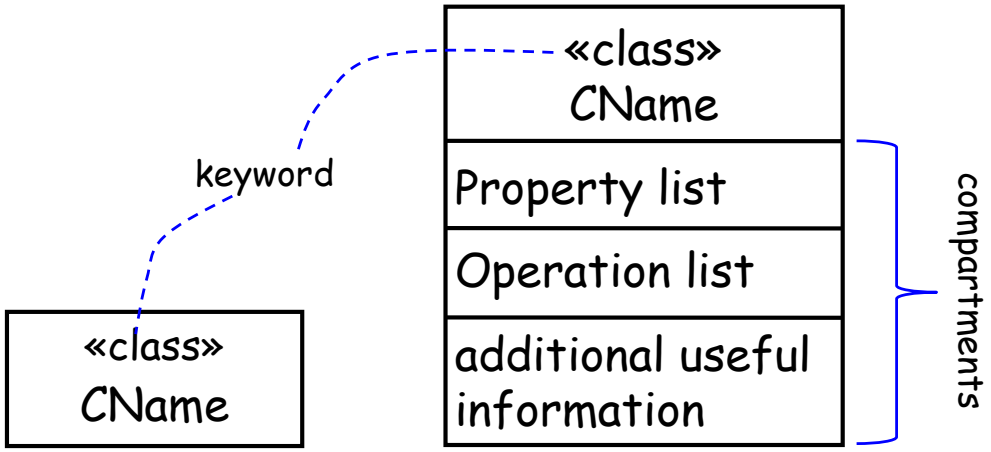
UML diagrams are comprised of model elements that are a semantic specification with a fixed notation. In accordance with *graph theory*, the elements are categorized according to vertex and edge. So, the most conducive way to learn UML is as you would any language: alphabet then word then sentence. Shape of a model element is analogous to alphabet, semantic of a model element is analogous to word, and combination of vertex and edge elements is analogous to sentence

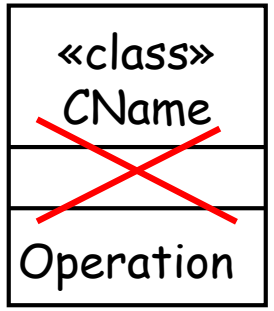
Two out of the 14 UML diagrams are relevant for architecture: Component and Communication

Class is key UML ModelElement

From course 2006, Class is a key/foundational UML ModelElement notationally

- Notation – box with keyword "class" enclosed by guillemets "« »" inside upper centre, followed below with centred name (given by modeler), and below that being compartments
- Auxiliary notes – there may be unlimited compartments including none, however there can be no empty compartment





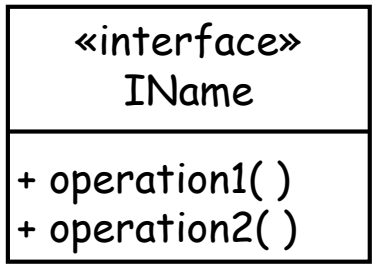
no empty compartment(s)

NANYANG TECHNOLOGICAL UNIVERSITY

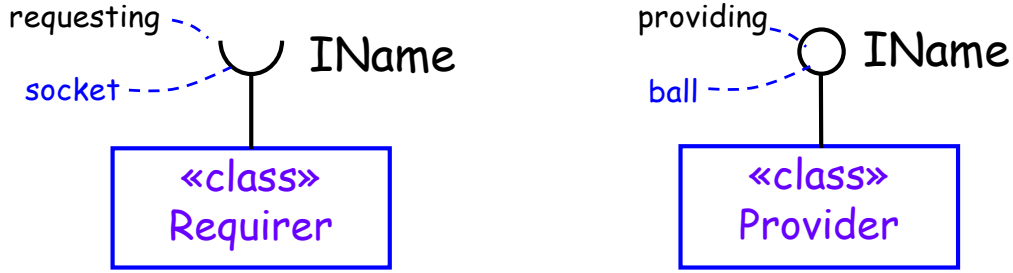
Remind me about Interface in OOP

From course 2002 "Object Oriented Programming", an Interface is a construct that declares to interested Classes that services they want are available, and secures those services from supplying Classes. In other words, the Interface isolates the requesting-services-Class from the providing-services-Class. An Interface is devoid of implementations of any kind

- Notation two forms, ie., *canonical* (longhand) and *elided* (shorthand)
  - canonical form – Class notation with mandatory keyword "interface", and below name, at least one compartment, the first of which contains all the Operations
  - elided form – ball and socket with name in proximity but no keyword, where the socket represents a Class' outgoing request for service, aka., *requesting*, and the ball represents the incoming order for service from the providing Class, aka., *providing*
- Auxiliary notes – although all the Operations are abstract, there is no additional notation to show so in canonical form because it is assumed the reader understands this about characteristic of the Interface



canonical form



elided/shorthand form

NANYANG TECHNOLOGICAL UNIVERSITY

Where does this fit?

aka., appreciating the big picture for today's lesson

Façade design pattern powers host component

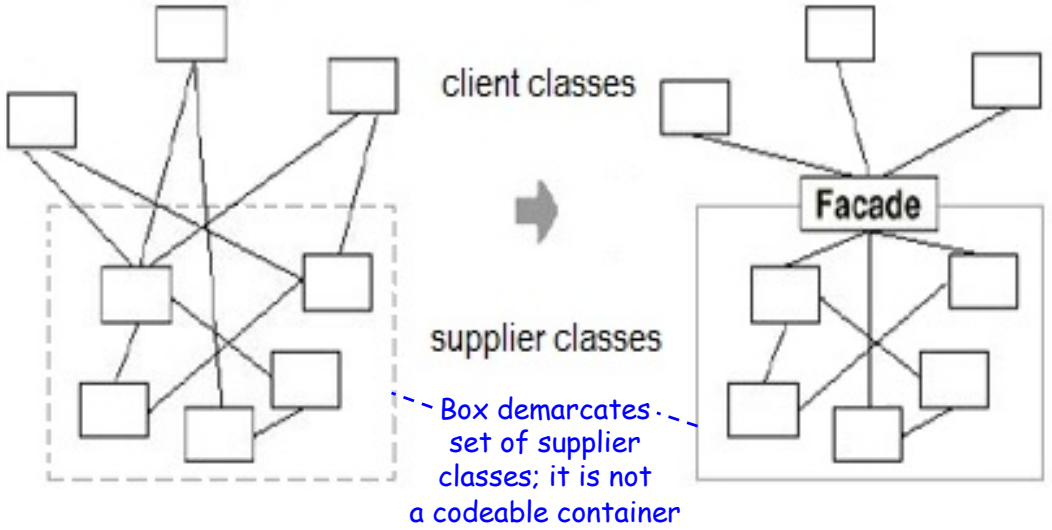
Sw architect considers only sw not hw port

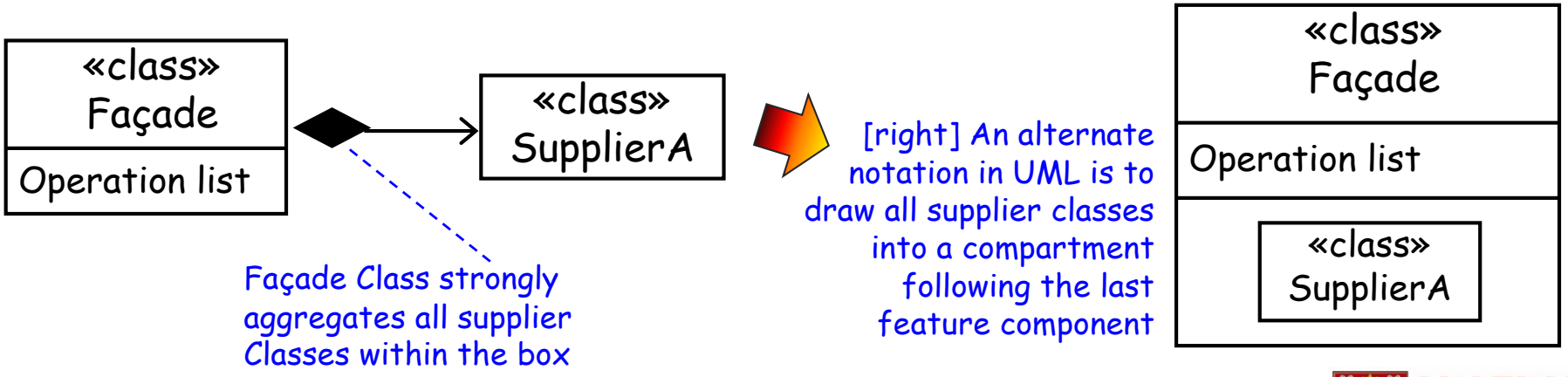
Ignoring network components

NANYANG TECHNOLOGICAL UNIVERSITY

Façade design ptrn powers host component

This highly-used design pattern offers a *single, unified implementation*, ie., "Façade" Class, through which external client classes can access a set of functions in supplier Classes [see right]. The purpose is to de-couple the supplying classes from direct access by the client, and by virtue of the Façade facilitate management, prioritization, and optimization of the services provision



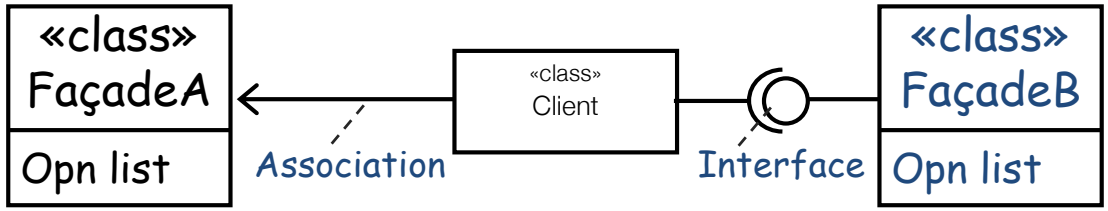


NANYANG TECHNOLOGICAL UNIVERSITY

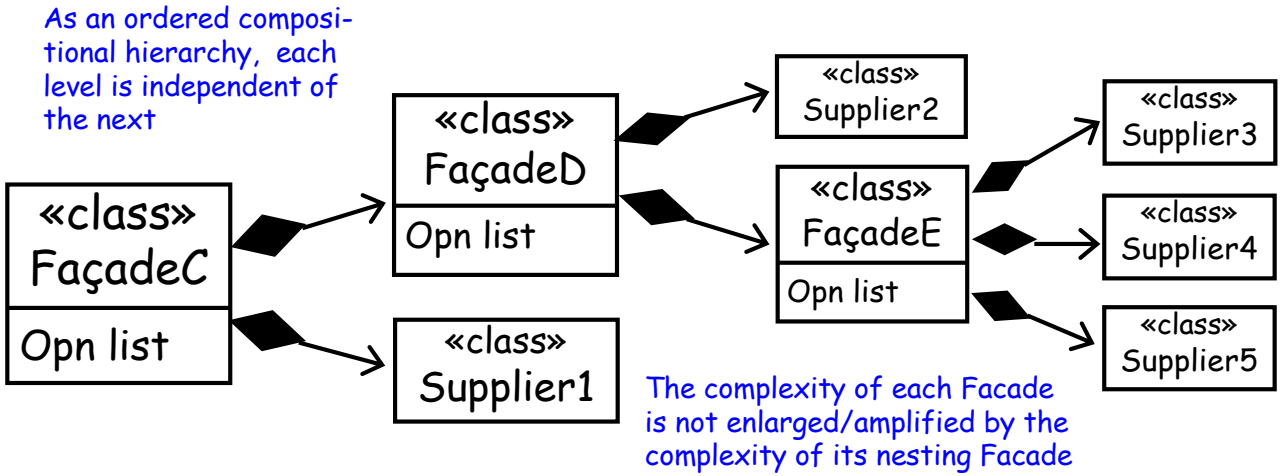
Façade design ptrn … host cmpnt (cont)

Façade Class' implementation does not override or repeat the functions of the supplier classes. Instead, Façade Class *delegates* the client class' calls to the appropriate supplier class, in the same way a *wrapper* does

Client classes may connect to the Façade Class by Association or Interface



Façade Class may strongly aggregate another Façade Class. This aggregation of another aggregation can continue without limit since the complexity is reduceable



NANYANG TECHNOLOGICAL UNIVERSITY

Sw architect considers only sw not hw port

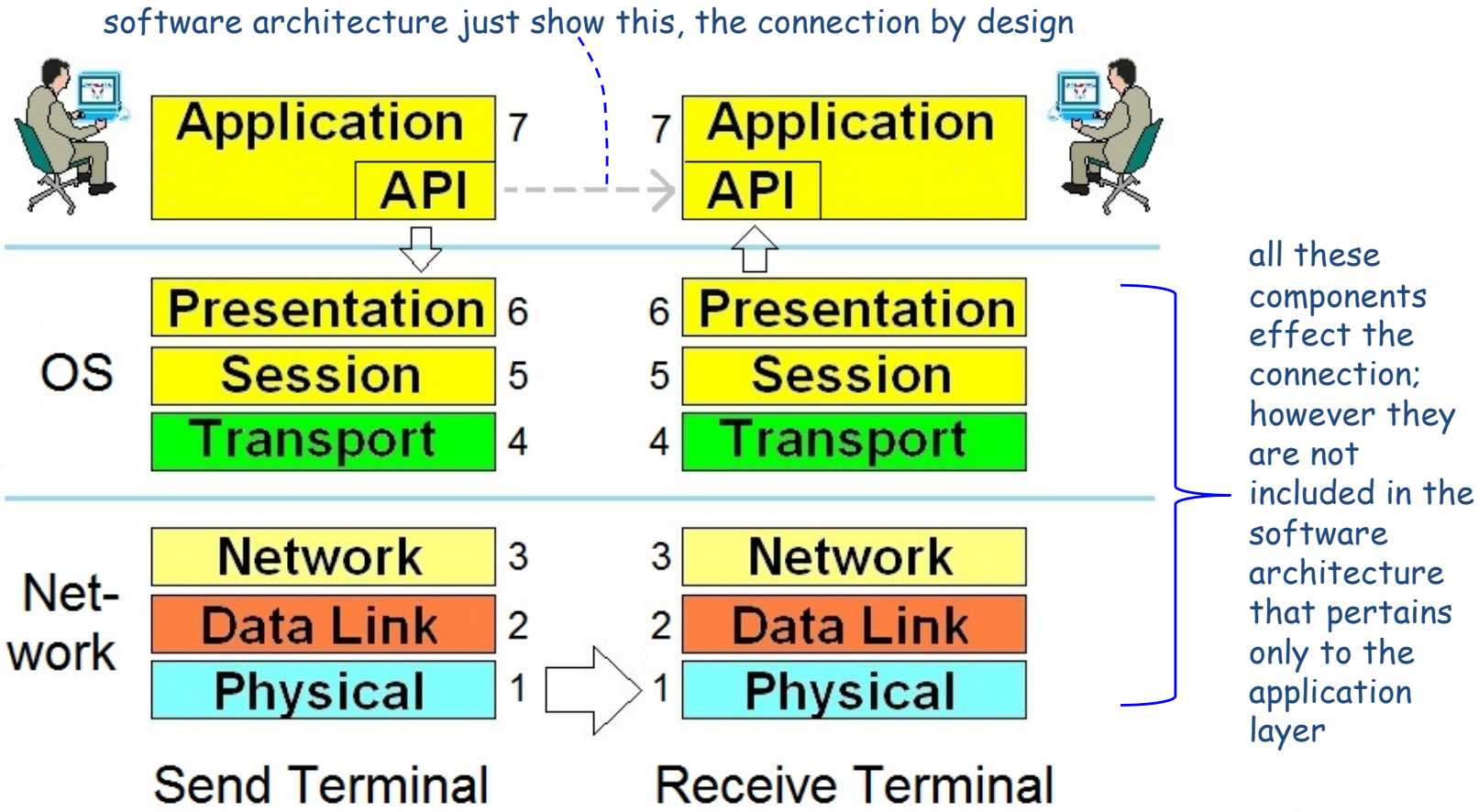
Hardware and software ports are substantially different

- A hardware port is a *physical* interface to a cable between the computer and peripheral, eg., keyboard, mouse, monitor, printer, speakers, webcam, or other computers or devices
- A software port is the endpoint of logical communication within the OS and the startpoint of the communication in a computer network. A software port is a *logical*, not physical, construct that identifies the application [recall 7 layers of software] or a specific service within the application installed in the computer owning the port. Finally, a software port is designated by a *port number* that codifies the IP address of the host computer and the protocol type of the connection

The second-most common mistake in developing the software architecture, after inclusion of hardware components, is the inclusion of hardware ports. Again, the architect needs to apply a *system perspective* [recall architectural thinking] as well as a solid understanding of software connections, to correctly apply ports into the software architecture

Ignoring network components

Network connections conform to the OSI model of seven layers of networking components: 1-3 are handled by network resources, and 4-6 are handled by the OS. However, it is only the cross-connection of 7th layer "application" that is of interest in the software architecture; the network components are ignored



You expect me to know *what* ?

aka., new knowledge in today's lesson

UML Component diagram = architecture diagram
UML Component = software component
UML Port = software port
UML Assembly and delegation Connectors
Connector + Interface = standard pairing
Port + standard pairing
Service, behavior, and complex Ports
Request-provide Interface guidelines
Three forms of Component diagram
Composition, decomposition, and specification
Frame is not a Component!

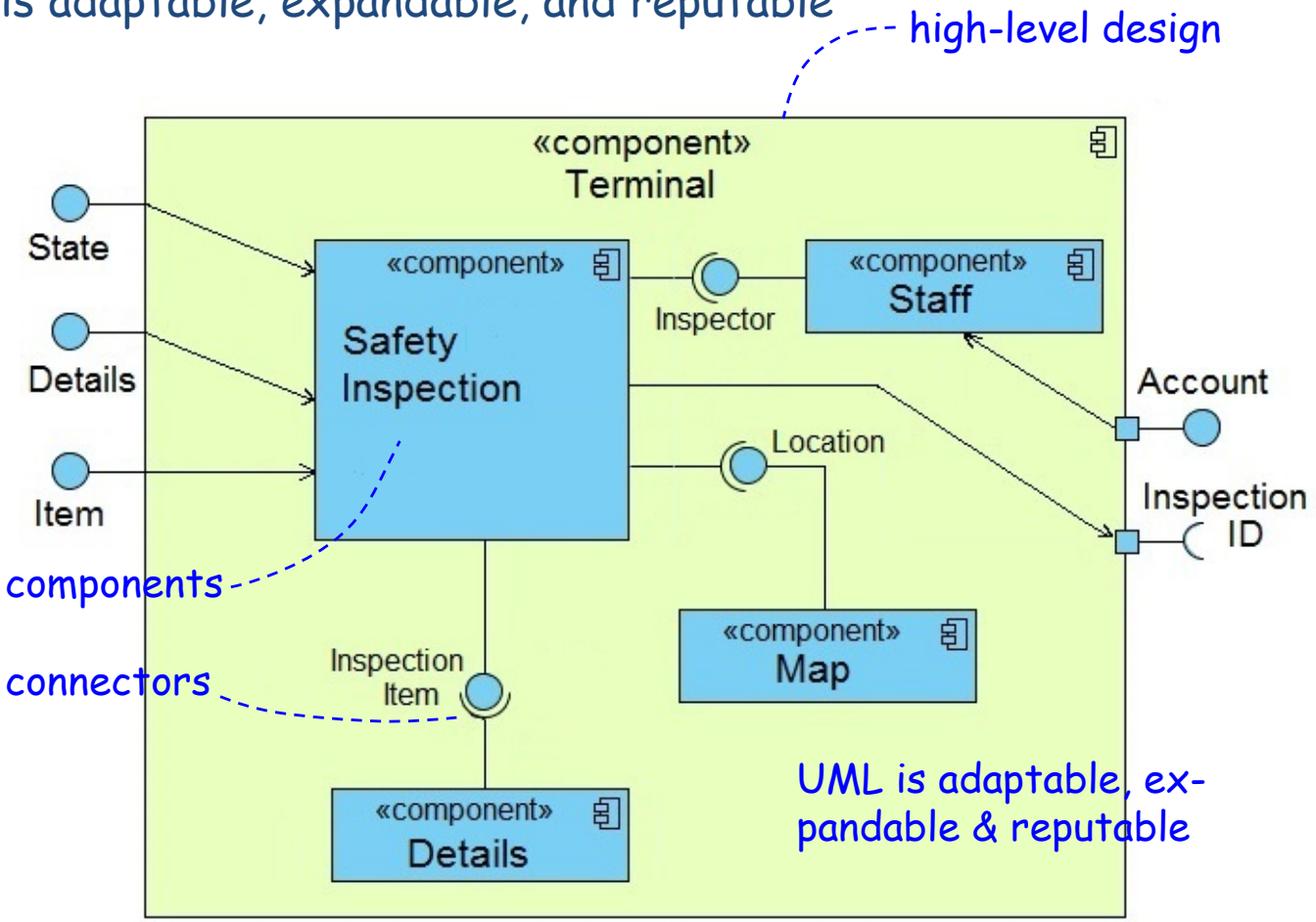
NANYANG TECHNOLOGICAL UNIVERSITY

Component diag = architecture structure

[recall architectural thinking] Software architecture is a high-level design of some software product or system. Accordingly, the software architect needs a modeling language that is adaptable, expandable, and reputable

[recall software components] The primary structural elements in a software architecture are components and connectors

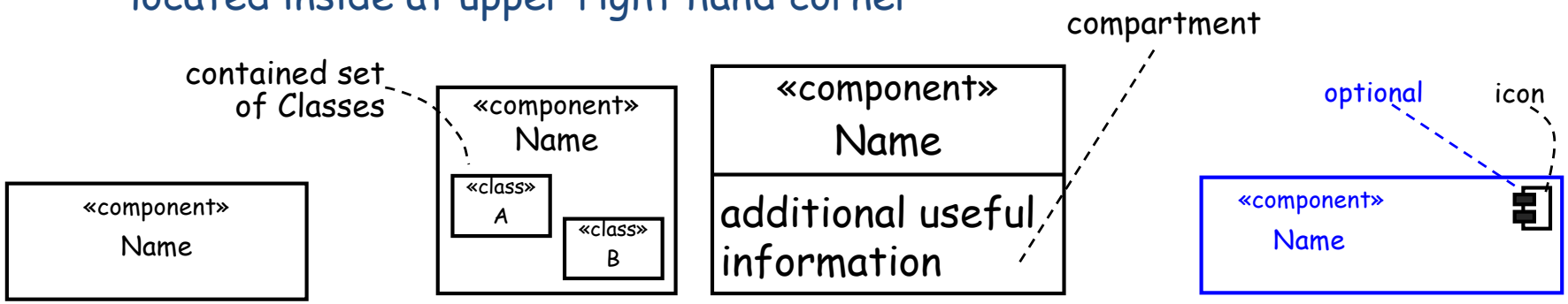
The UML Component diagram brings all these together, making the UML Component diagram ideal for modeling software architecture structure



UML Component = software component

[Recall software component] The UML Component models the software component thoroughly; it is a *container* of a 'set of classes'

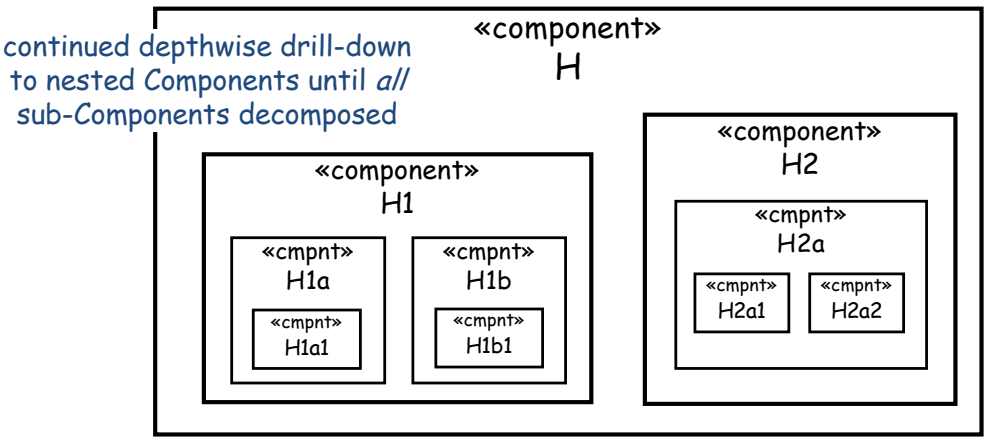
- Notation – Class notation with mandatory keyword "component", and below name, optional compartments containing informative details deemed by the modeler useful to provide. Owned members (Classes) of the Component are drawn inside the top-level box with the keyword and name
- Auxiliary notes – optional icon may be shown in Component to assist in recognition of Component ModelElement. Icon's notation is box with wings (older UML version notation, now superseded, of Component) located inside at upper right hand corner

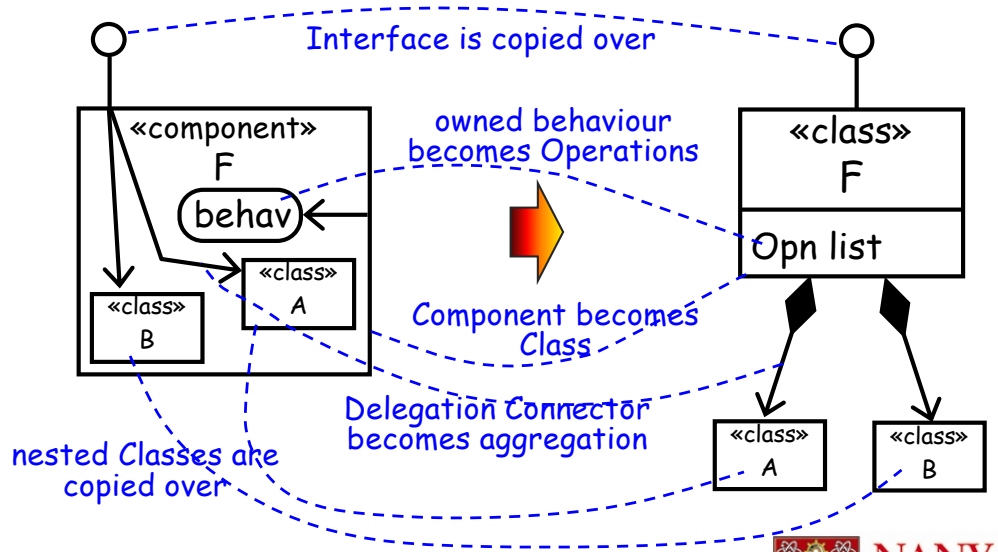


UML Component = sw component (cont)

Classes contained in a Component can be either "a big ball of mud" or arranged into one or more sub-Components. The corollary is a Component may *nest* other Components, to an unlimited depth in an ordered composition-al hierarchy [see upper right]

Importantly, the host Compo-nent must *own* behavioural feat-ures in order to delegate calls incoming/outgoing apropos its owned members. The most straightforward explanation is: the Façade design pattern *manifests* the host Component [see lower right]

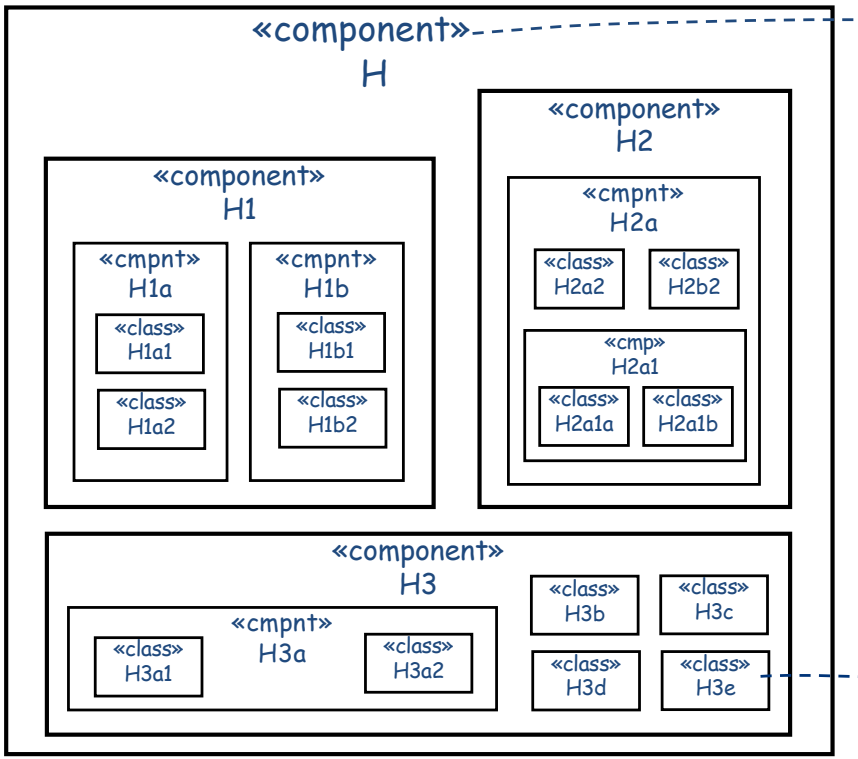




NANYANG TECHNOLOGICAL UNIVERSITY

UML Component = sw component (cont)

Combining the UML Component's unlimited decomposability with manifestation by the Façade design pattern, facilitates *estimation* of (minimum) *number* of Classes that are signified by the architecture diagram, eg., … [see below]



every hosting Component manifests into a Façade Class

Component	Façade Class	Nested Class
H	H	
H1	H1	
H1a	H1a	H1a1, H1a2
H1b	H1b	H1b1, H1b2
H2	H2	
H2a	H2a	H2a2, H2a3
H2a1	H2a1	H2a1a, H2a1b
H3	H3	H3b, H3c, H3d, H3e
H3a	H3a	H3a1, H3a2

every nested Class moved over

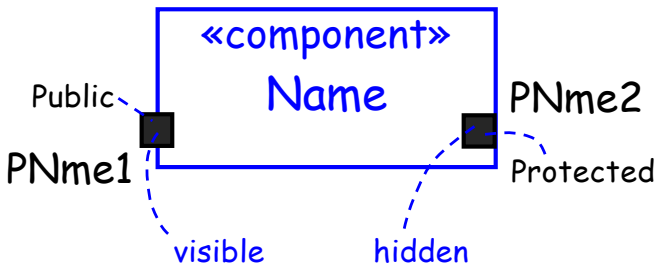
Total minimum of Classes
for Component diagram = 23

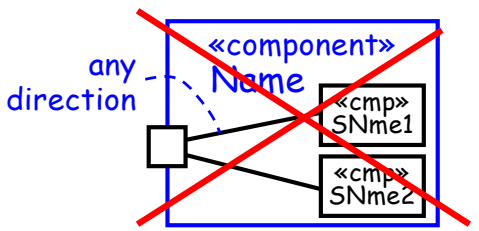
NANYANG TECHNOLOGICAL UNIVERSITY

UML Port = software port

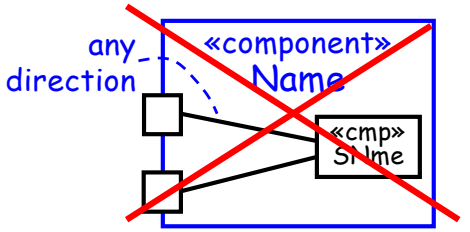
This is a Property that specifies a *distinct interaction point* between a Component and its environment. In software context, this is an *address* of the Component connected into a computer network. If the software component is not operating in a network, then Port is not used

- Notation – box with name in proximity, straddling the Component box or inside it. The box is substantially smaller than the Component box
- Auxiliary notes
  - Port inside the Component means it is 'Protected' visibility, and when straddling it is 'Public'
  - if a Component sends messages arriving at its Ports to internal sub-Components, one Port serves only one sub-Component, and one sub-Component can only be served by one Port





1 Port → 2 Cmp No!



2 Port → 1 Cmp No!

NANYANG TECHNOLOGICAL UNIVERSITY

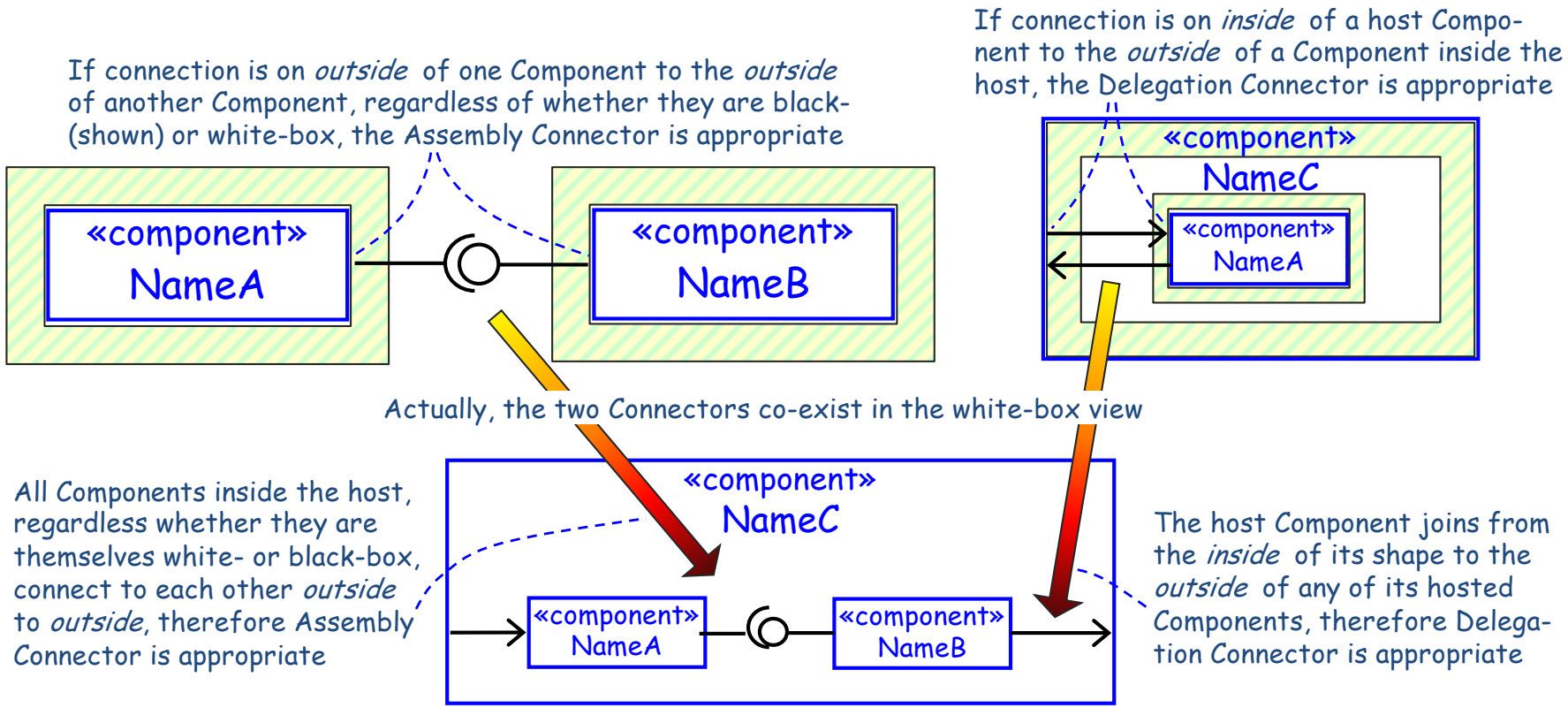
UML Assembly & Delegation Connector

This is an information exchange channel between Components, as simple as a pointer or as complex as a network session. There are two kinds of connectors: Assembly and Delegation. An *Assembly* Connector is an Interface [per course 2002] between Components, qualifying them as provider or requester of services. A *Delegation* Connector is a forwarder of service requests inside a Component, either entering (external call to a host Component is "pushed in" to an internal realizing sub-Component) or exiting (call generated by an internal sub-Component is "pushed out" to the host Component for external transmittal)

- Notation is different for each kind of connector
  - Assembly – ball-and-socket (eluded notation) shown close together practically touching, or separate but joined by a dashed open-arrowed line (to notionally join them) towards provider from requester
  - Delegation – solid open-arrowed line in direction of the service call
- Auxiliary notes – Assembly already paired with Interface. Delegation connector needs pairing with the external requester or provider of service Interface; from that the Delegation derives its direction

UML Assembly & Delegation … (cont)

Often it is said that Assembly Connector is for black-box view, and Delegation Connector is for white-box view. However, it is more accurate to say Connector usage is fixed by where its ends attach to the Components [see below]

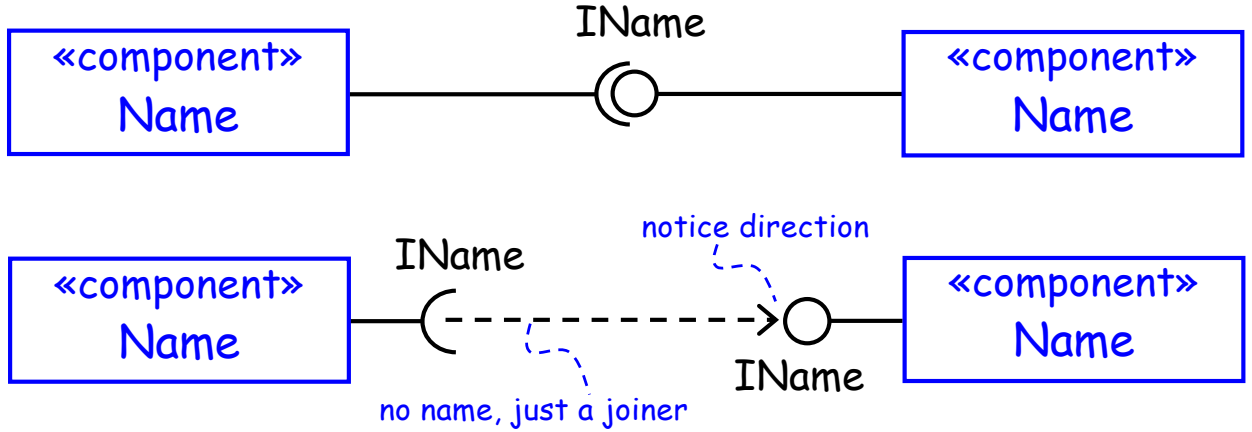


NANYANG TECHNOLOGICAL UNIVERSITY

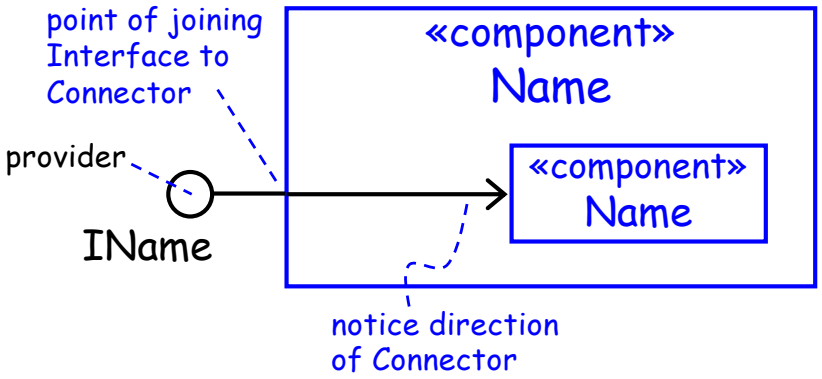
Connector + Interface = std pairing

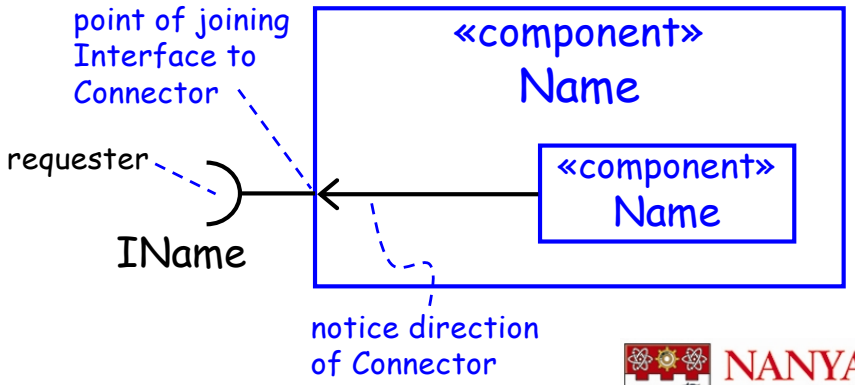
These are standard arrangements/pairings of Connectors with Interfaces

These two examples are equivalent in meaning; the dashed line is just a readability aid



Delegation Connector

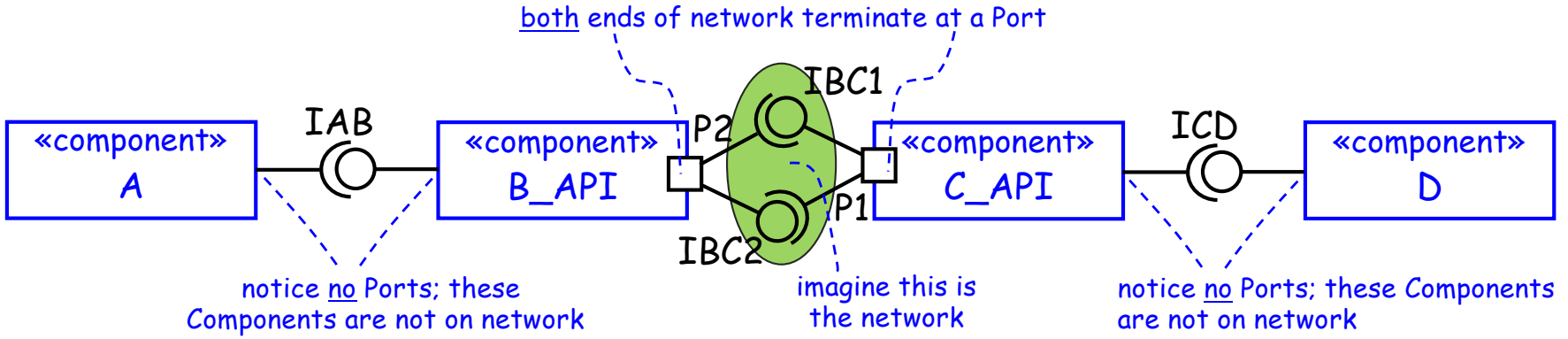


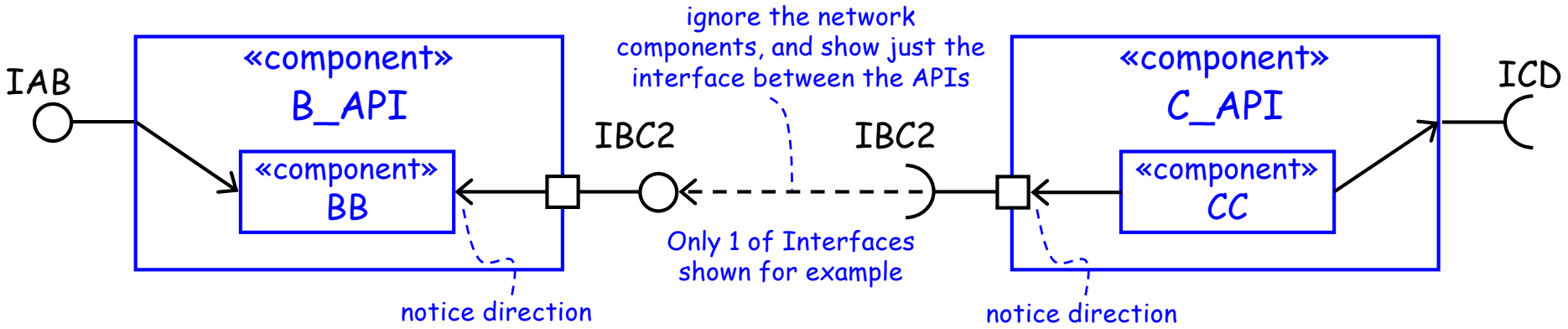


NANYANG TECHNOLOGICAL UNIVERSITY

Port + standard pairing

If the architecture involves connection to a computer network, then Ports will be shown on those only Components actually/physically joined to the network





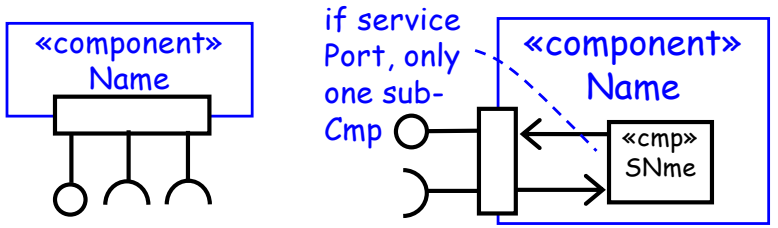
NANYANG TECHNOLOGICAL UNIVERSITY

Service, behavior, and complex Ports

A host Component containing sub-Components is analogous to the Façade design pattern, and like the Façade Class, a host Component either has:

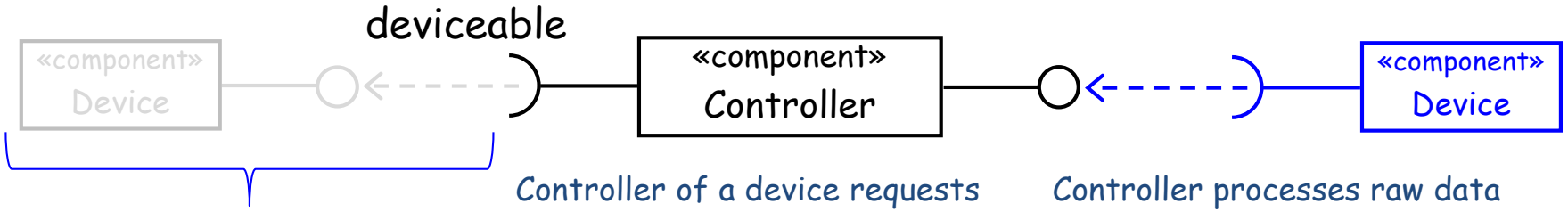
- no integral behaviour and offers just the services of its sub-Components. A Port on this kind of Component is a *Service* Port
- integral behaviour that is offered in addition to its sub-Components' services. A Port on this kind of Component is a *Behaviour* Port

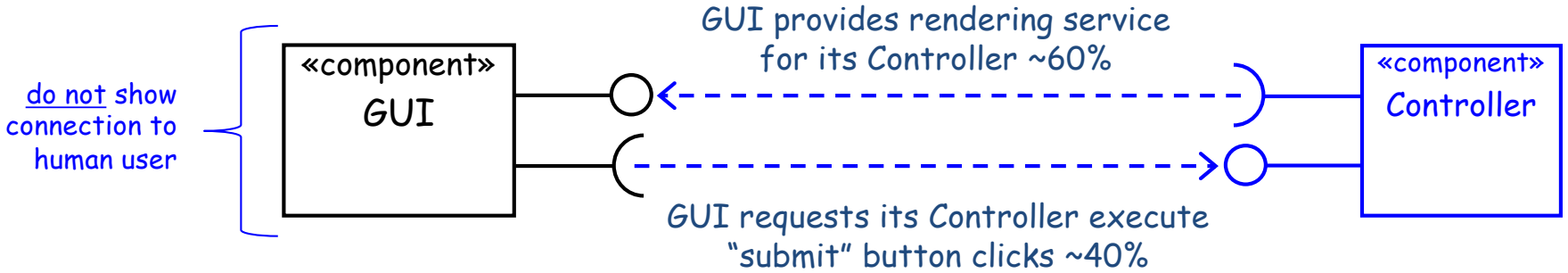
Finally, requesting and providing Interfaces may congregate on one Port [see right]; the owning Component exposes a complex service offering to external Components



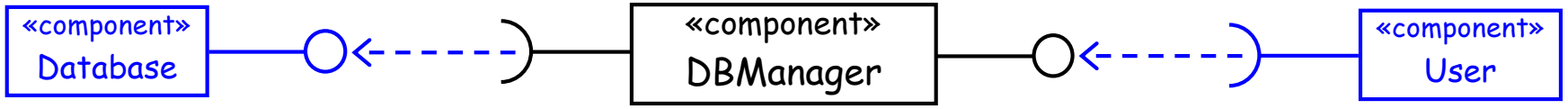
NANYANG TECHNOLOGICAL UNIVERSITY

Request-provide Interface guidelines





do show database in architecture diagram



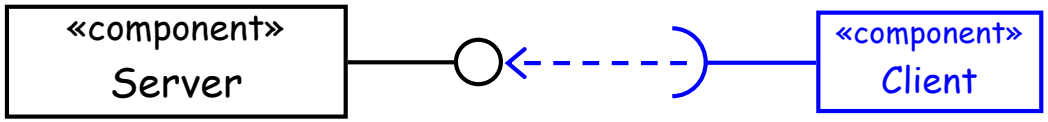
DBManager requests its database to provide inherent data processing service, e.g., write, fetch, index, etc

DBManager provides translation and priority to users upon their requests

NANYANG TECHNOLOGICAL UNIVERSITY

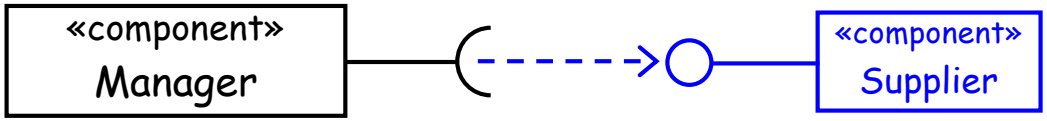
Req-prov Interface guidelines (cont)

Server are self-sufficient and typically do not request for anything from other components

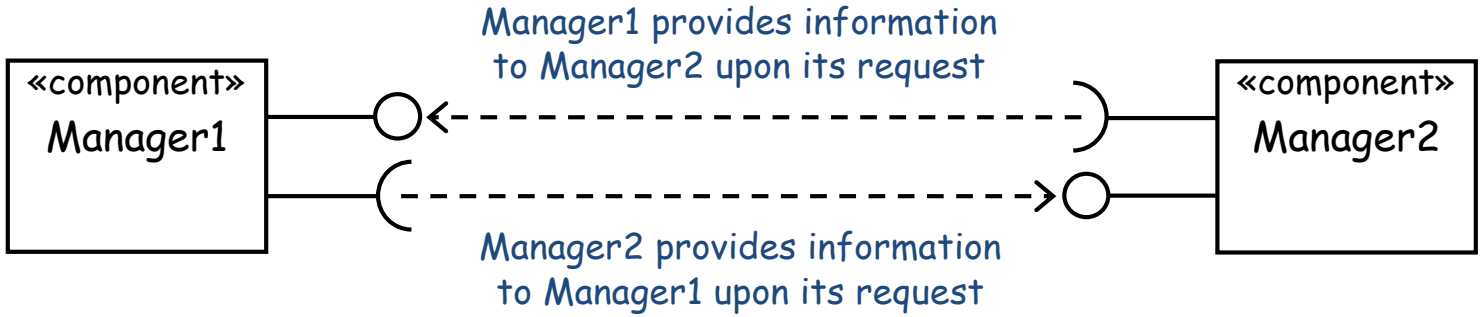


Server provides service to its clients upon their requests

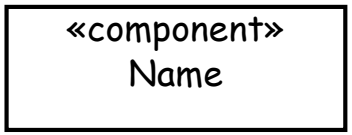
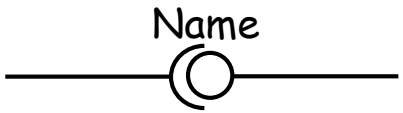
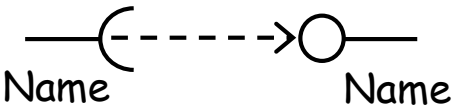
Manager does not provide its Suppliers with output/feedback



Manager requests its Suppliers to do work that they are designed to do



Summary of UML ModelElements comprising Component diagram

ModelElement	Notation
Component	
Port	
Requiring Interface	
Providing Interface	
Assembly Connector	
Assembly Connector	
Delegation Connector	
Dependency	

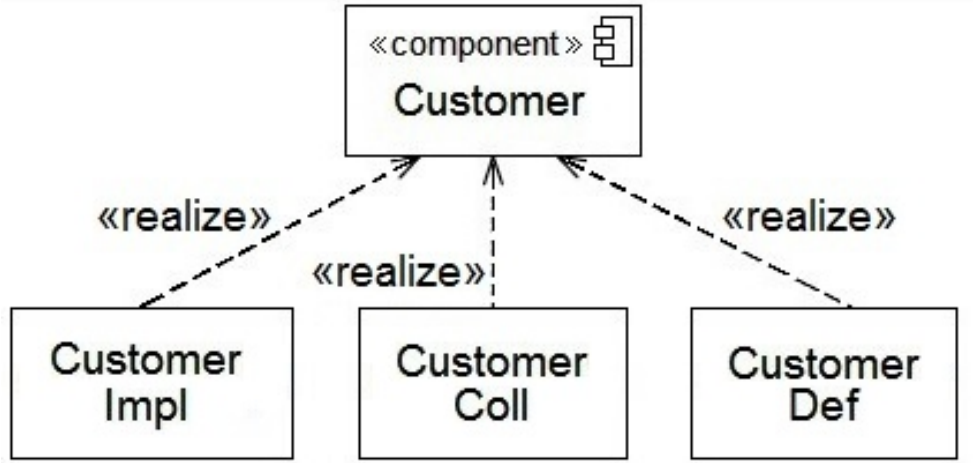
Three forms of Component diagram

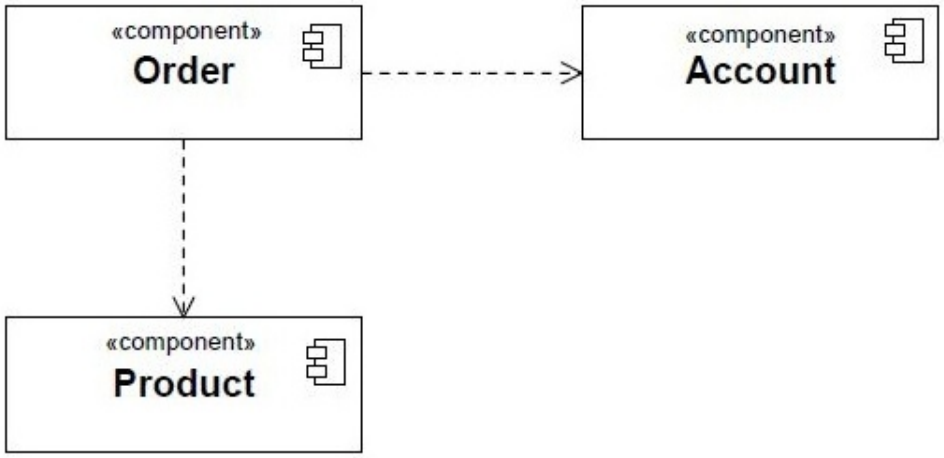
The UML Component diagram is a purposeful arrangement of notationally-belong-ing ModelElements to specify a particular configuration of Components in a software architecture of interest. There are three forms according to usage of the Component diagram

1. Specification – design-centric mapping of Components to other Model-Elements. This is '*to design*' [recall arch thinking] various relationships possible for the software before drawing the architecture. Typically, the vertices are black-box view
2. Composition – breadthwise expansion as new Components are appended to an existing aggregate. This is used to assemble the architecture in a *bottom-up* strategy. Typically, the vertices are black-box view
3. Decomposition – depthwise drill-down as Components are broken down into sub-Components and so on, ie., *top-down* strategy. This is used to build detail about the internals of select Components in an architecture. The uppermost (from the depthwise perspective) vertices are white-box view, while the lowermost are black-box view

Specification form of diagram

[see right] This diagram is 'to design', the realization of the "Customer" Component's services by three sub-Components "CustomerImpl", "CustomerColl", & "CustomerDef". This models the architect's early design ideas for the decomposition architecture



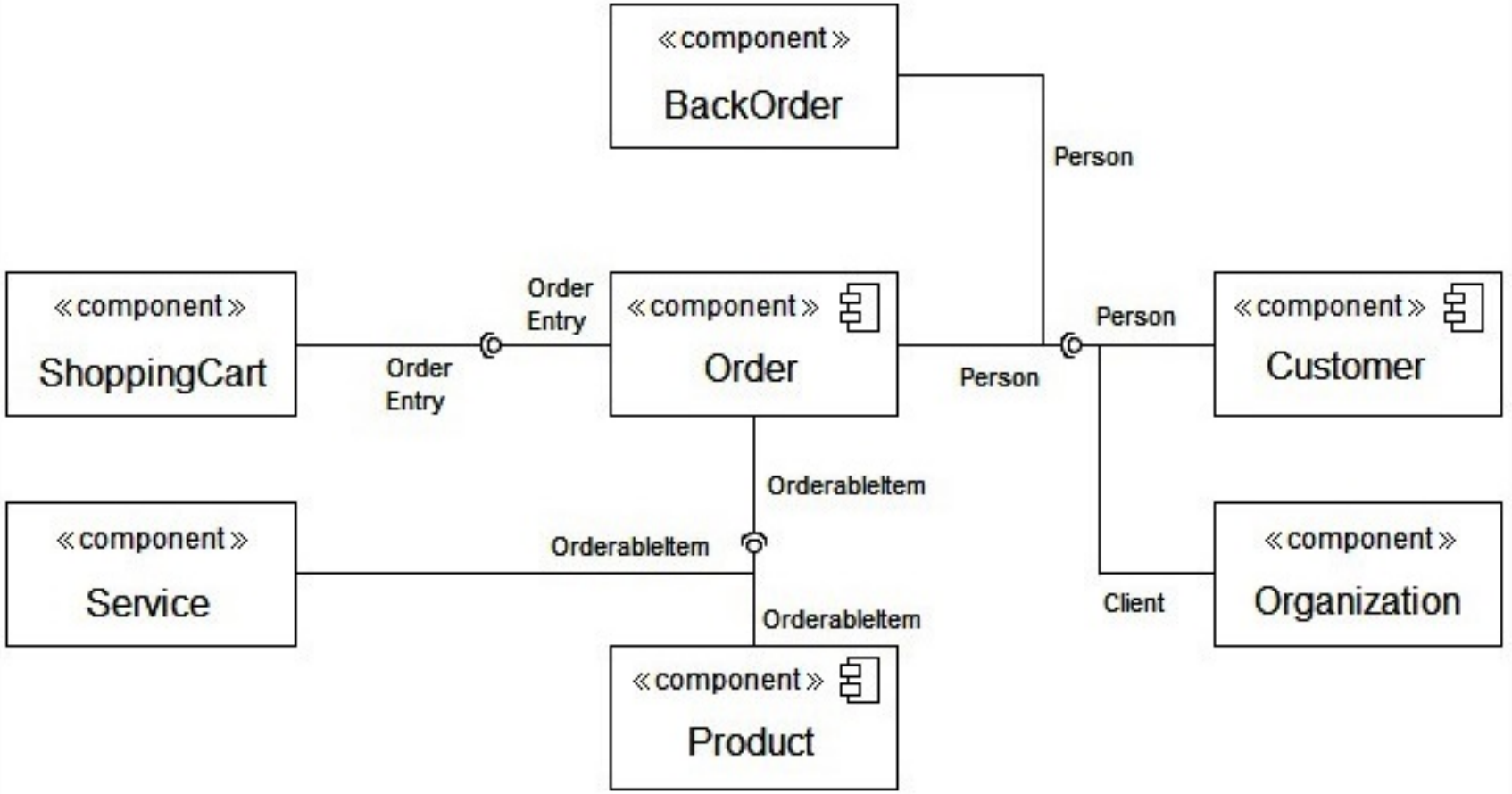


[see left] This diagram shows the *general Dependency* that the "Order" Component has with Components "Product" & "Account". This models the architect's early design ideas for the composition architecture

NANYANG TECHNOLOGICAL UNIVERSITY

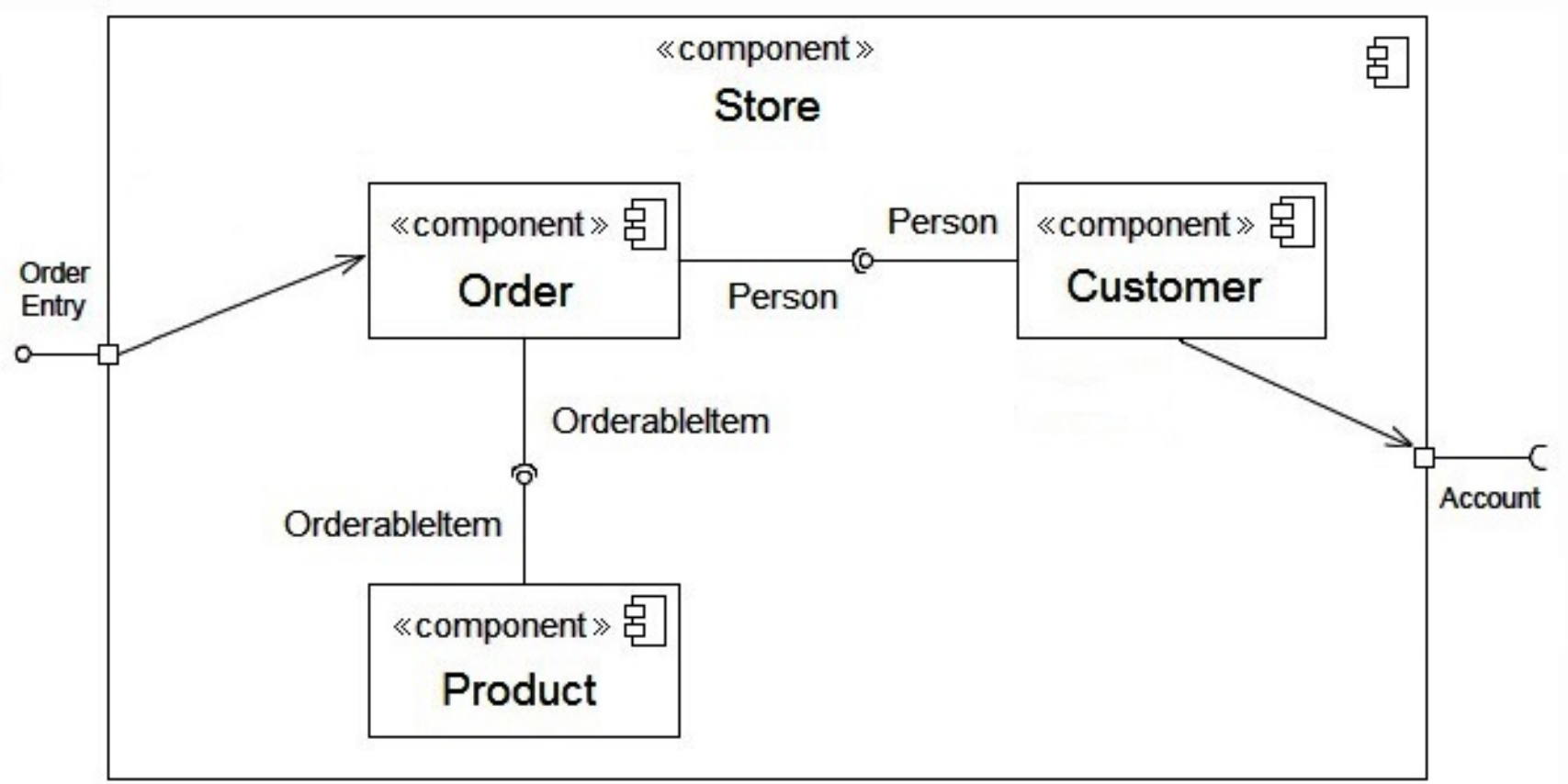
Composition form of diagram

Imagine being asked to draw the architecture for a software. You would piece together the Components like a puzzle, one Component at a time, until all the Components have been connected into the aggregate. For this example, there probably are external connections; they just haven't yet been added



Decomposition form of diagram

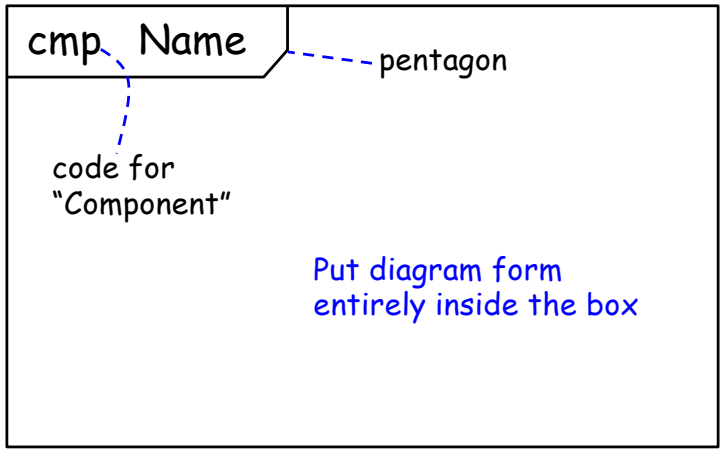
Imagine being asked to draw the internal sub-Components for a Component. You would start with the Interfaces of the outer Component and work inwards. These outer Interfaces come from the composition diagram on the previous page. The icons are random [I personally don't use the icon in my diagrams]



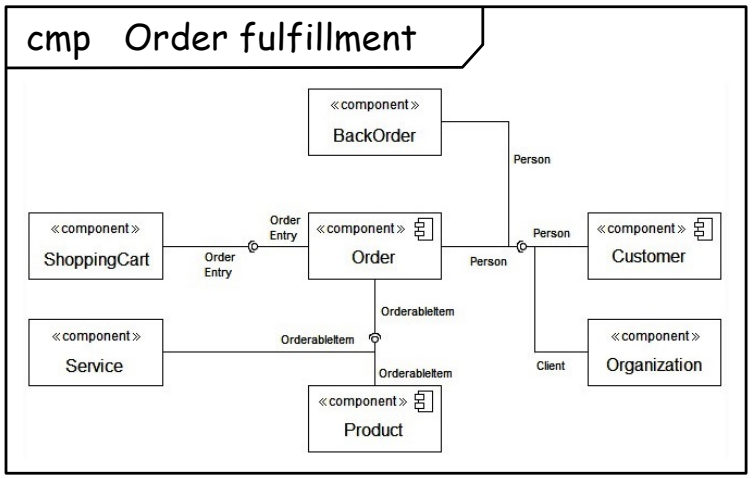
Frame is not a Component!

This is a combination of *border* surrounding and *caption* of the diagram; it is devoid of semantics, and in particular, it is not a Component! It may be used for all three forms of the diagram; notwithstanding its use is optional

- Notation – a box outline with a pentagon, containing the diagram's caption, inside touching the box at its upper left corner [see below]
- Auxiliary notes – caption format is "diagram code diagram name" where code for Component diagram is "cmp", and name is as assigned



Example of frame with composition diagram



NANYANG TECHNOLOGICAL UNIVERSITY

Practice makes perfect

aka., mini- workshop

Task: Draw Component diagram for ADDER

NANYANG TECHNOLOGICAL UNIVERSITY

Draw Component diagram for ADDER

[Recall workshops in previous lessons for software components]

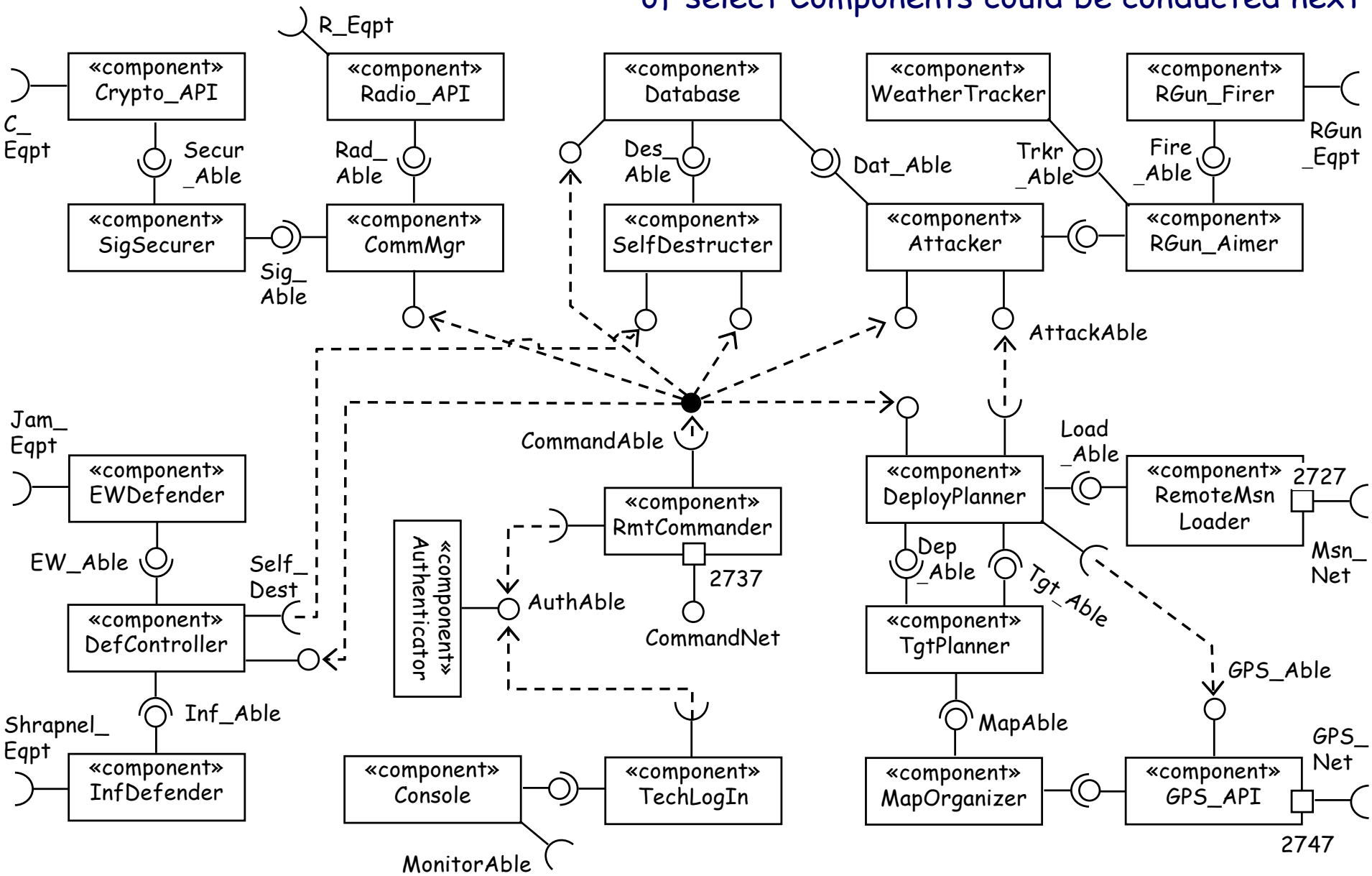
NANYANG TECHNOLOGICAL UNIVERSITY

Doing step #3 in architectural process

Drawing the architecture's formal model is step #3 in *jumpstart* [recall architectural process]. For this, the software architect has full discretion in choosing the *modeling language* [recall architectural thinking]. With the architect going with UML, its distinctive schemes will be imbedded into his/her design, eg., service requesting and providing, inter-component interfaces, delegation to sub-components, ports, and three diagram forms. Assuming the first two steps are finished and the initial architecture is completed for the ADDER, the following is a suggestion of how to conduct step#3

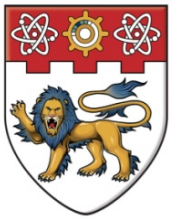
1. Draw Component box for all software components
2. Pair components to attach requesting or providing Interface on each, until all components have an Interface
3. Add Ports to API Components
4. Aggregate supporting Components with Facades and/or respective managers, if any
5. Assign names to all ModelElements not yet named

This is composition form of the UML Component diagram. Grouping or decomposition of select Components could be conducted next



What did I just do? aka., learning wrap-up

- UML Component diagram is the architecture diagram
- Component = software component
- Port = software port
- Assembly and Delegation Connectors
- Connector + Interface = standard pairing
- Port + standard pairing
- Service, behavior, and complex Ports
- Three forms of Component diagram:
  - Specification, composition, and decomposition
- Frame is not a Component!



NANYANG TECHNOLOGICAL UNIVERSITY

Modeling Software Structure using UML

presented by

Shar Lwin Khin
Research Scientist
SCSE
lkshar@ntu.edu.sg
N4-02c-76

Courtesy of Kevin Anthony Jones' slides