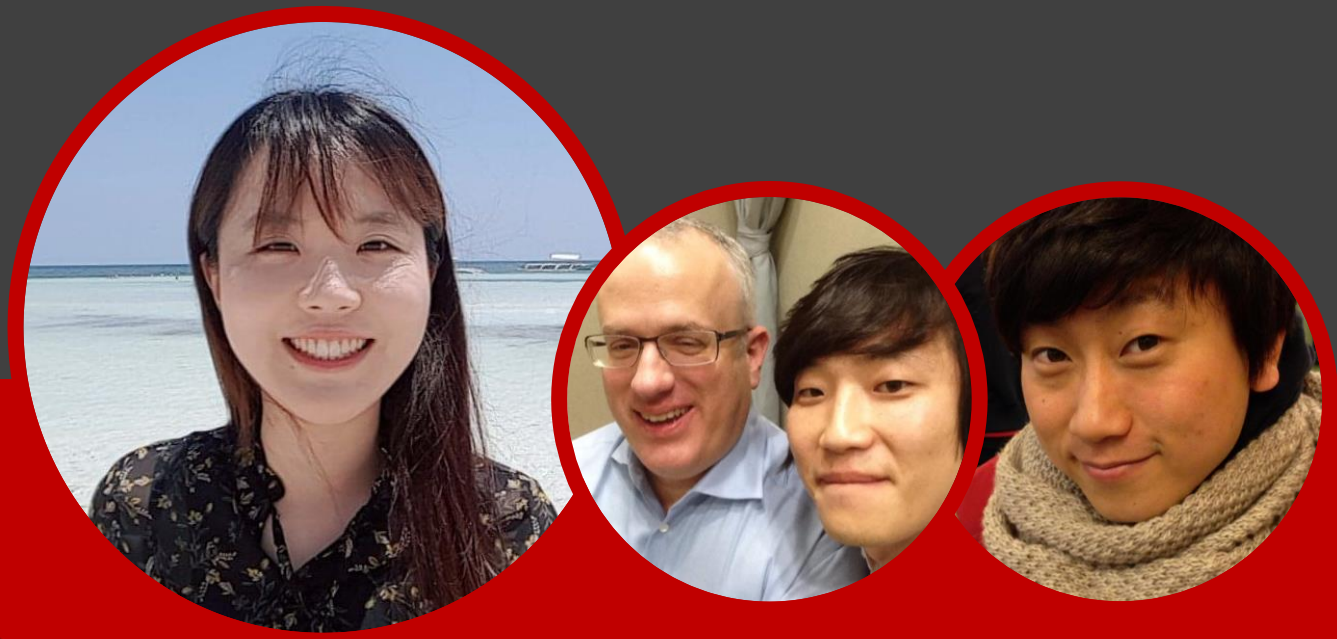
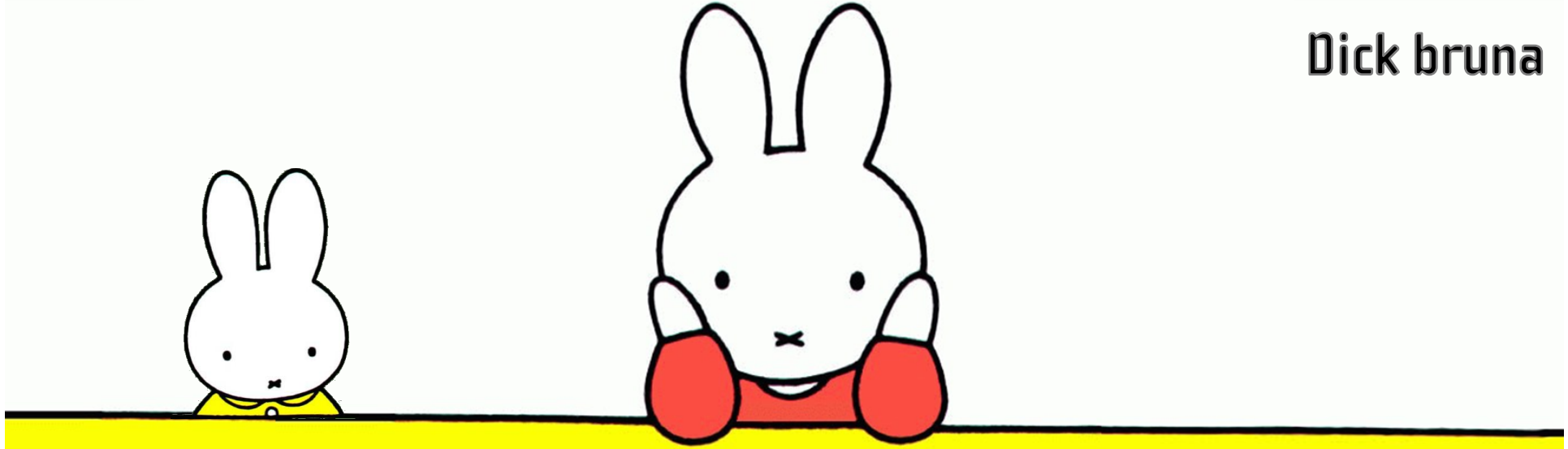
한빛미디어 HanbitMedia, Inc.

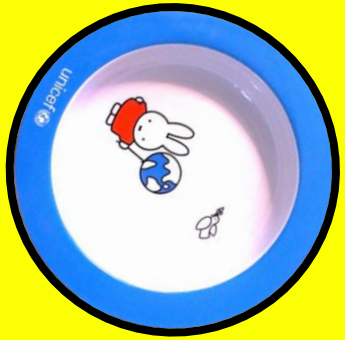
Bsidesoft co.




1998

Dick bruna





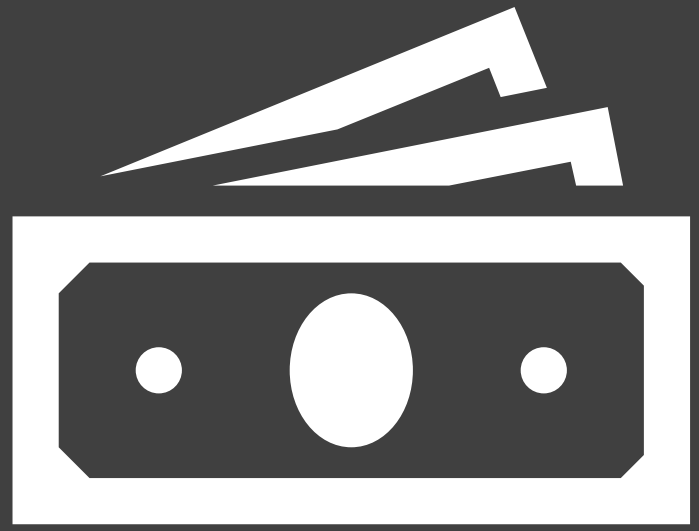
CODE  SPITZ

83

OBJECT

1 2 3 4 5 6

motivation



Philosophy

Philosophy

Value Principle

Philosophy

Value

Principle

Pattern

Philosophy

Value Principle Xoriented

Pattern

Philosophy

Relativism : 토마스 쿤(과학혁명의 구조)
Rationalism : 러커토시 임레(수학적 발견의 논리: 증명과 반박)
난장판 : 파울 파이어아벤트(방법에의 도전)과 그 이후

Value Principle Xoriented

Pattern

Philosophy

Relativism : 토마스 쿤(과학혁명의 구조)
Rationalism : 러커토시 임레(수학적 발견의 논리: 증명과 반박)
난장판 : 파울 파이어아벤트(방법에의 도전)과 그 이후

Value Principle Xoriented

Pattern

Philosophy

Relativism : 토마스 쿤(과학혁명의 구조)
Rationalism : 러커토시 임레(수학적 발견의 논리: 증명과 반박)
난장판 : 파울 파이어아벤트(방법에의 도전)과 그 이후

Value Principle Xoriented

Pattern

Philosophy

Relativism : 토마스 쿤(과학혁명의 구조)
Rationalism : 러커토시 임레(수학적 발견의 논리: 증명과 반박)
난장판 : 파울 파이어아벤트(방법에의 도전)과 그 이후

Value Principle Xoriented

Communication
Simplicity
Flexibility

Pattern

Philosophy

Relativism : 토마스 쿤(과학혁명의 구조)
Rationalism : 러커토시 임레(수학적 발견의 논리: 증명과 반박)
난장판 : 파울 파이어아벤트(방법에의 도전)과 그 이후

Value

Communication
Simplicity
Flexibility

Principle

Local consequences
Minimize repetition
Symmetry
Convention

Xoriented

Pattern

Philosophy

Relativism : 토마스 쿤(과학혁명의 구조)
Rationalism : 러커토시 임레(수학적 발견의 논리: 증명과 반박)
난장판 : 파울 파이어아벤트(방법에의 도전)과 그 이후

Value

Communication
Simplicity
Flexibility

Principle

Local consequences
Minimize repetition
Symmetry
Convention

Xoriented

OOP : SOLID, DRY..
Reactive
Functional
..

Pattern

Abstraction

Abstration

Generalization : 일반화 – modeling, function, algorithm
Association : 연관화 – reference, dependence
Aggregation : 집단화 – group, category

Abstration

Generalization : 일반화 – modeling, function, algorithm
Association : 연관화 – reference, dependence
Aggregation : 집단화 – group, category

Abstration

Generalization : 일반화 – modeling, function, algorithm
Association : 연관화 – reference, dependence
Aggregation : 집단화 – group, category

Abstration

Generalization : 일반화 – modeling, function, algorithm
Association : 연관화 – reference, dependence
Aggregation : 집단화 – group, category

Data Ab Procedural Ab

Abstration

Generalization : 일반화 – modeling, function, algorithm
Association : 연관화 – reference, dependence
Aggregation : 집단화 – group, category

Data Ab Procedural Ab OOP Ab

Abstration

Generalization : 일반화 – modeling, function, algorithm
Association : 연관화 – reference, dependence
Aggregation : 집단화 – group, category

Data Ab Procedural Ab OOP Ab

Modeling
Categorization
Grouping

Abstration

Generalization : 일반화 – modeling, function, algorithm
Association : 연관화 – reference, dependence
Aggregation : 집단화 – group, category

Data Ab

Modeling
Categorization
Grouping

Procedural Ab

Generalization
Capsulization

OOP Ab

Abstration

Generalization : 일반화 – modeling, function, algorithm
Association : 연관화 – reference, dependence
Aggregation : 집단화 – group, category

Data Ab

Modeling
Categorization
Grouping

Procedural Ab

Generalization
Capsulization

OOP Ab

Generalization
Realization
Dependency
Association
Directed Association
Aggregation
Composition

Timing

Program & Timing

Program & Timing

LANGUAGE CODE

Program & Timing

LANGUAGE CODE

MACHINE LANGUAGE

Program & Timing

LANGUAGE CODE

MACHINE LANGUAGE

FILE

Program & Timing

LANGUAGE CODE

MACHINE LANGUAGE

FILE

LOAD

Program & Timing

LANGUAGE CODE

MACHINE LANGUAGE

FILE

LOAD

RUN

Program & Timing

LANGUAGE CODE

MACHINE LANGUAGE

FILE

LOAD

RUN

TERMINATE

Program & Timing

LANGUAGE CODE — LINT TIME

MACHINE LANGUAGE

FILE

LOAD

RUN

TERMINATE

Program & Timing

LANGUAGE CODE — LINT TIME

MACHINE LANGUAGE — COMPILE TIME

FILE

LOAD

RUN

TERMINATE

Program & Timing

LANGUAGE CODE — LINT TIME

MACHINE LANGUAGE — COMPILE TIME

FILE

LOAD

RUN — RUN TIME

TERMINATE

Script Program

Script Program

LANGUAGE CODE — LINT TIME

MACHINE LANGUAGE — COMPILE TIME

FILE

LOAD

RUN — RUN TIME

TERMINATE

Script Program

LANGUAGE CODE — LINT TIME

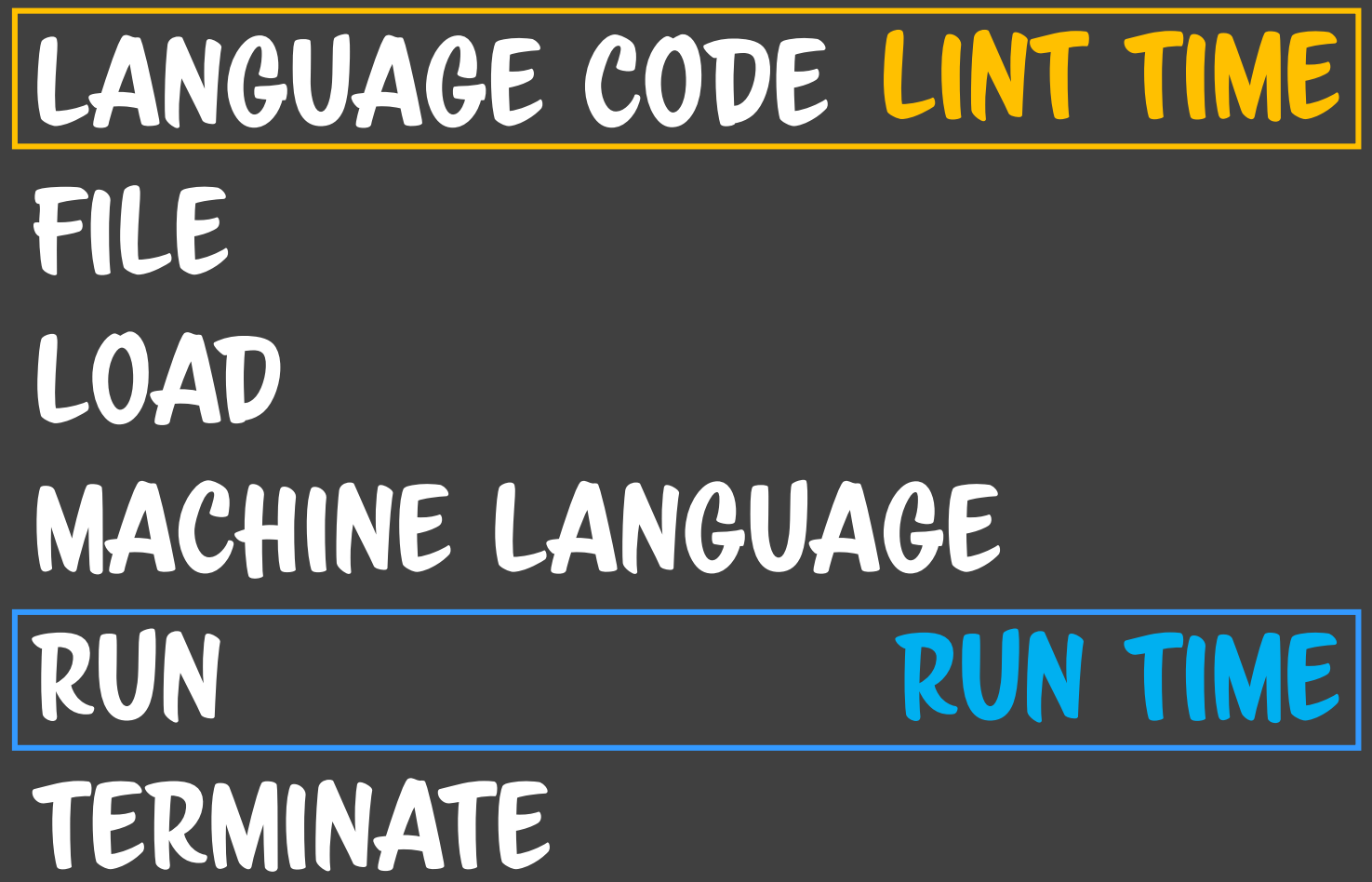
MACHINE LANGUAGE — COMPILE TIME

FILE

LOAD

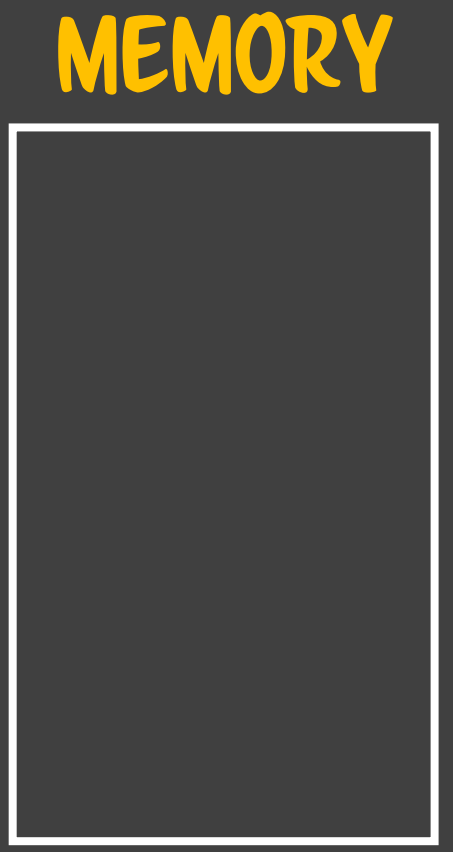
RUN — RUN TIME

TERMINATE



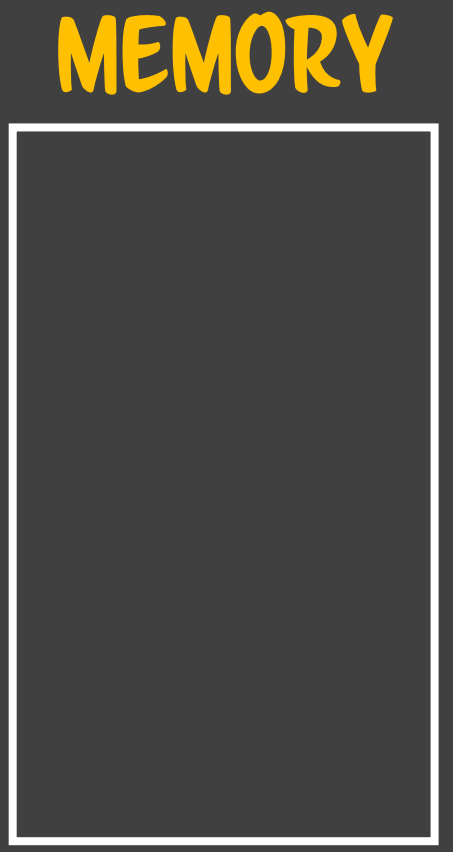
Runtime

Runtime



Runtime

LOADING



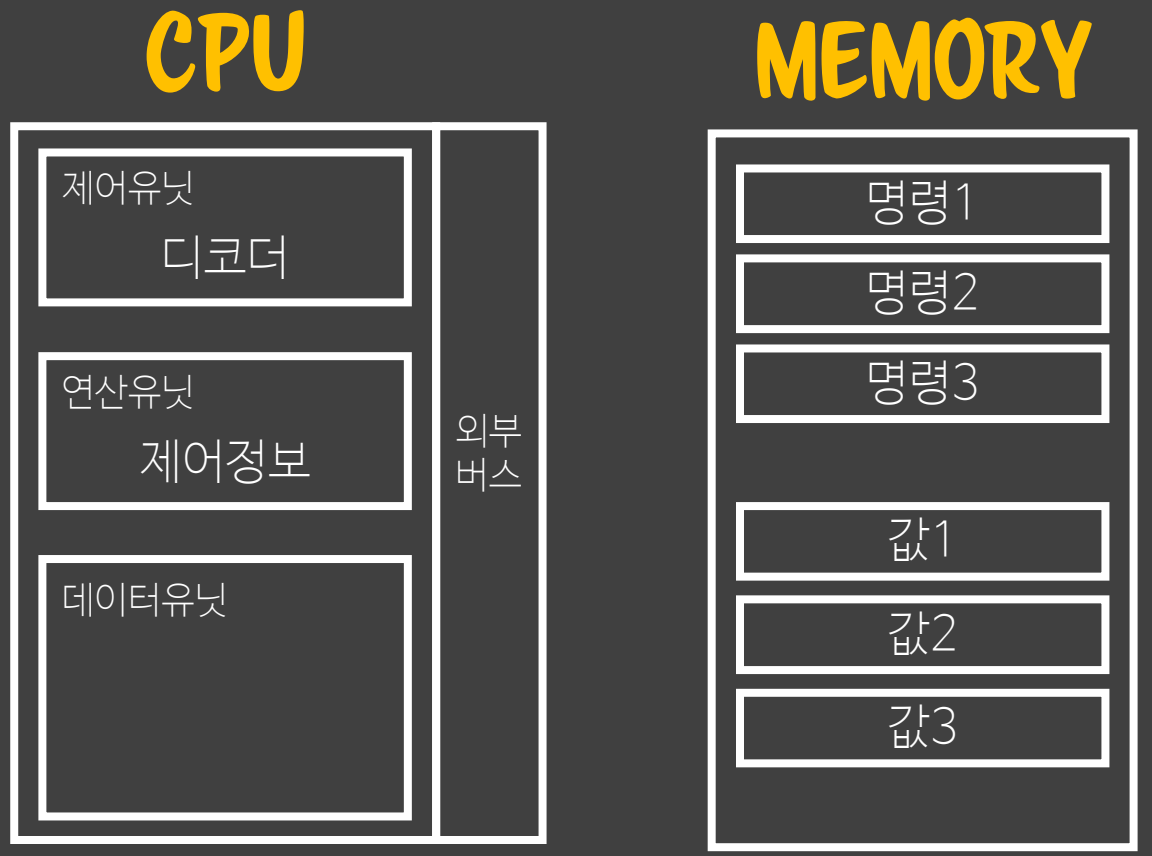
Runtime

LOADING

Runtime

LOADING

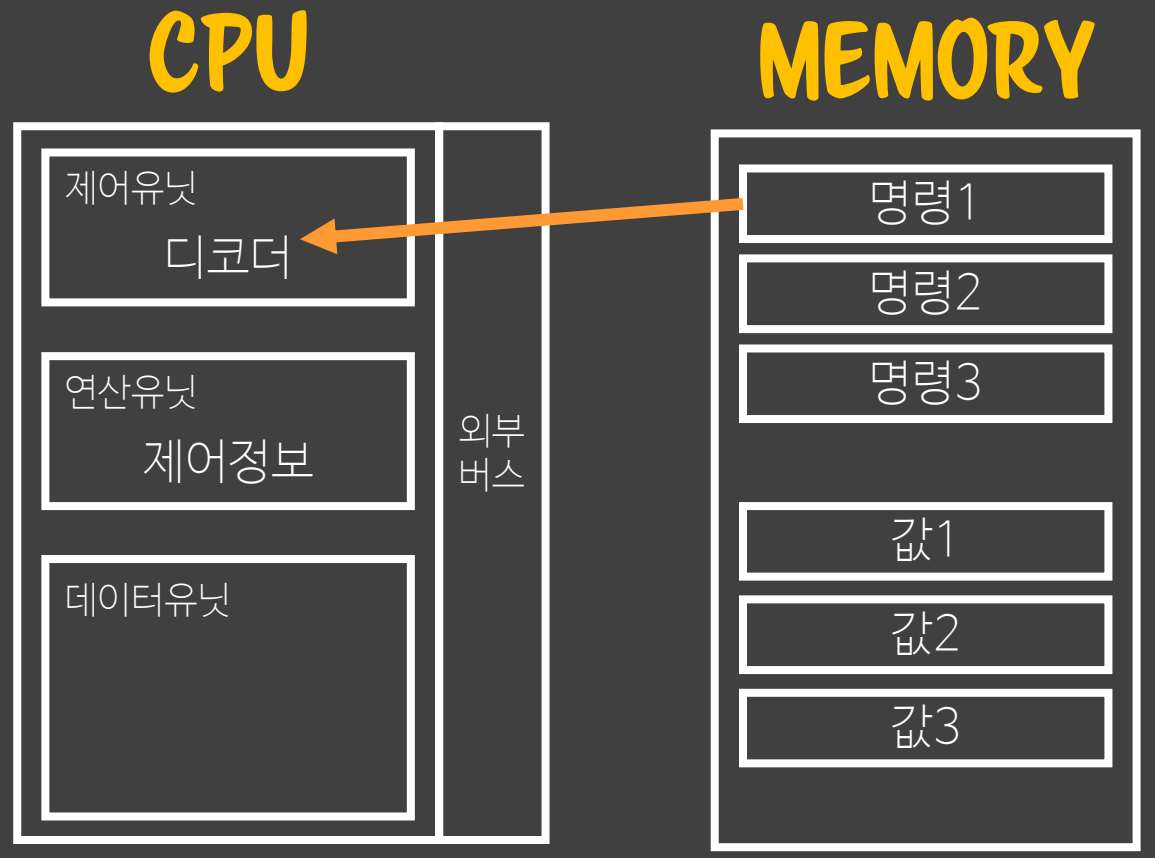
INSTRUCTION FETCH & DECODING



Runtime

LOADING

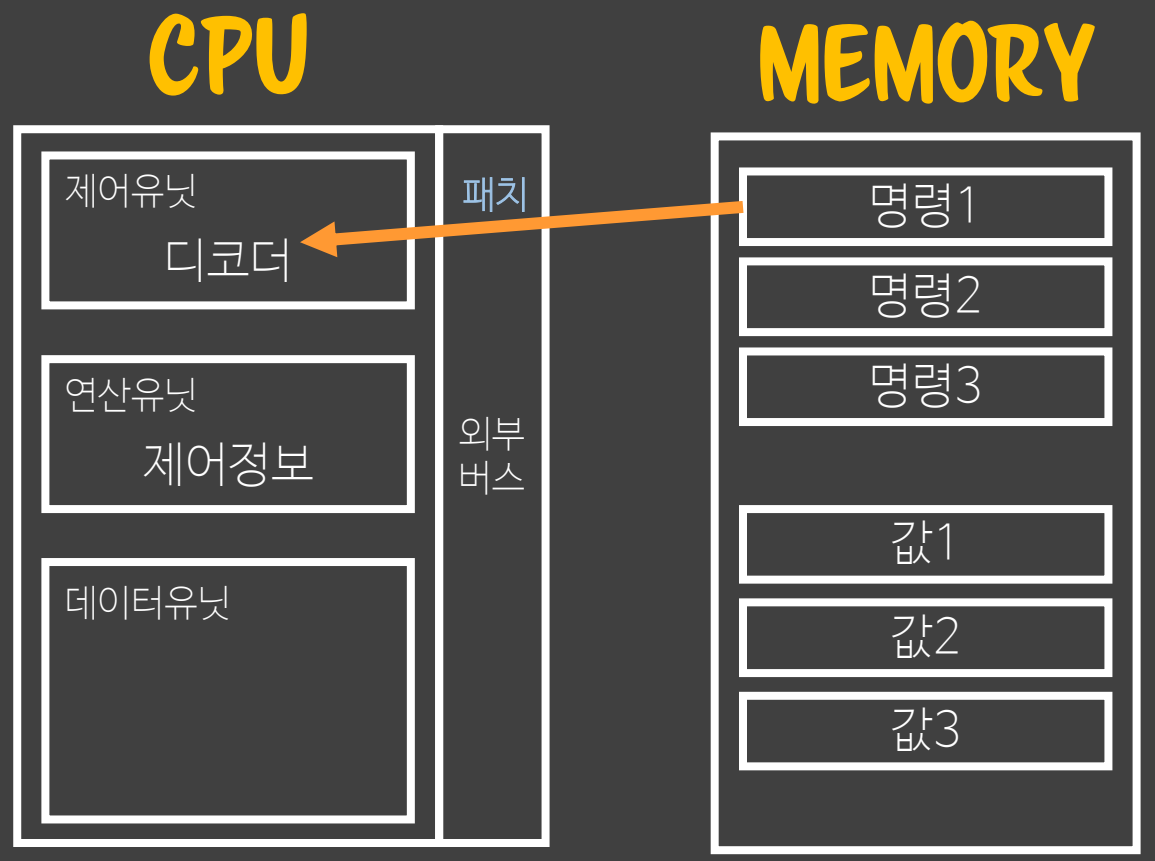
INSTRUCTION FETCH & DECODING



Runtime

LOADING

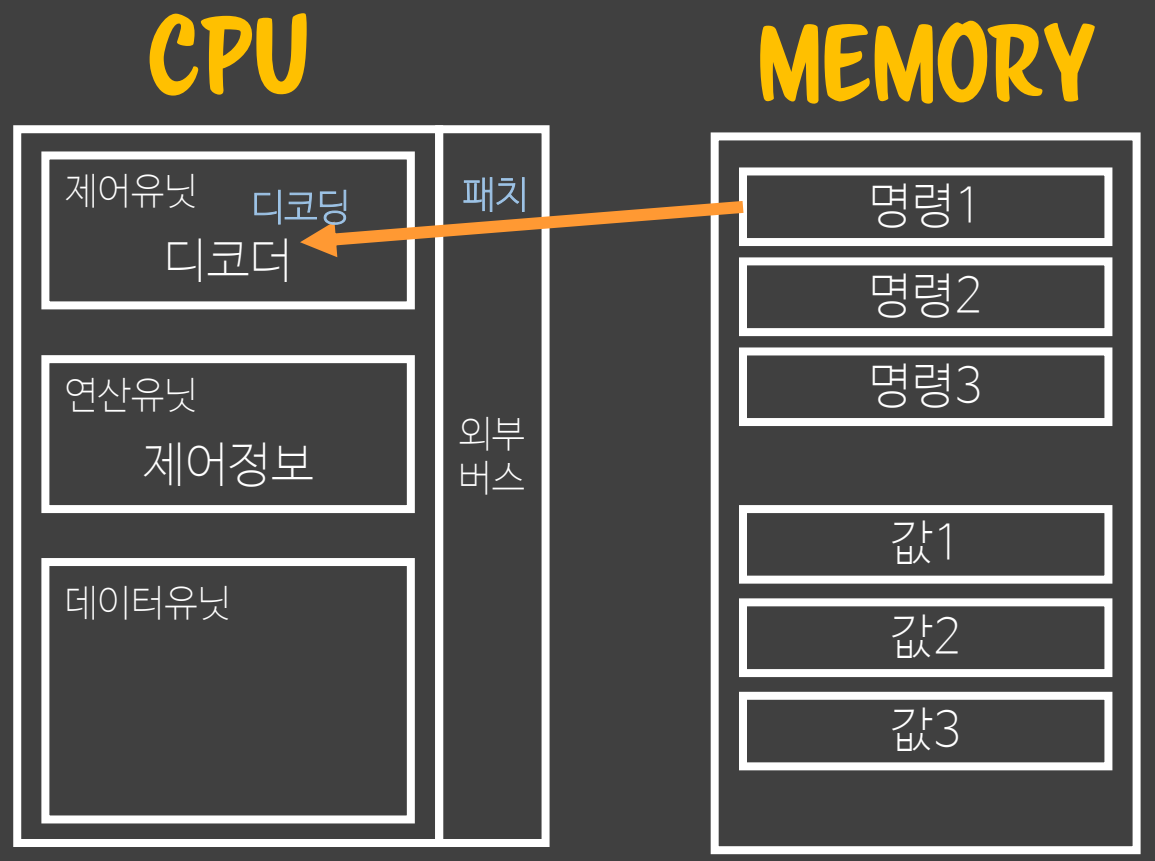
INSTRUCTION FETCH & DECODING



Runtime

LOADING

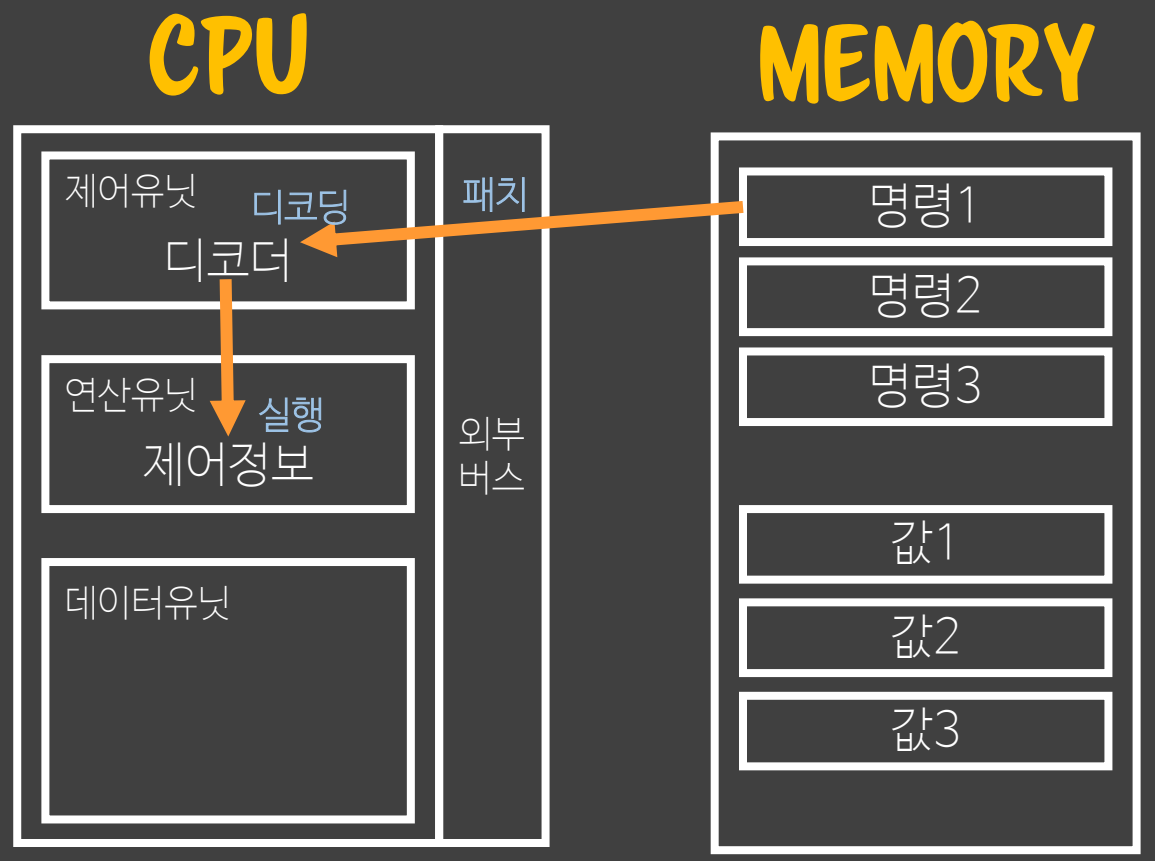
INSTRUCTION FETCH & DECODING



Runtime

LOADING

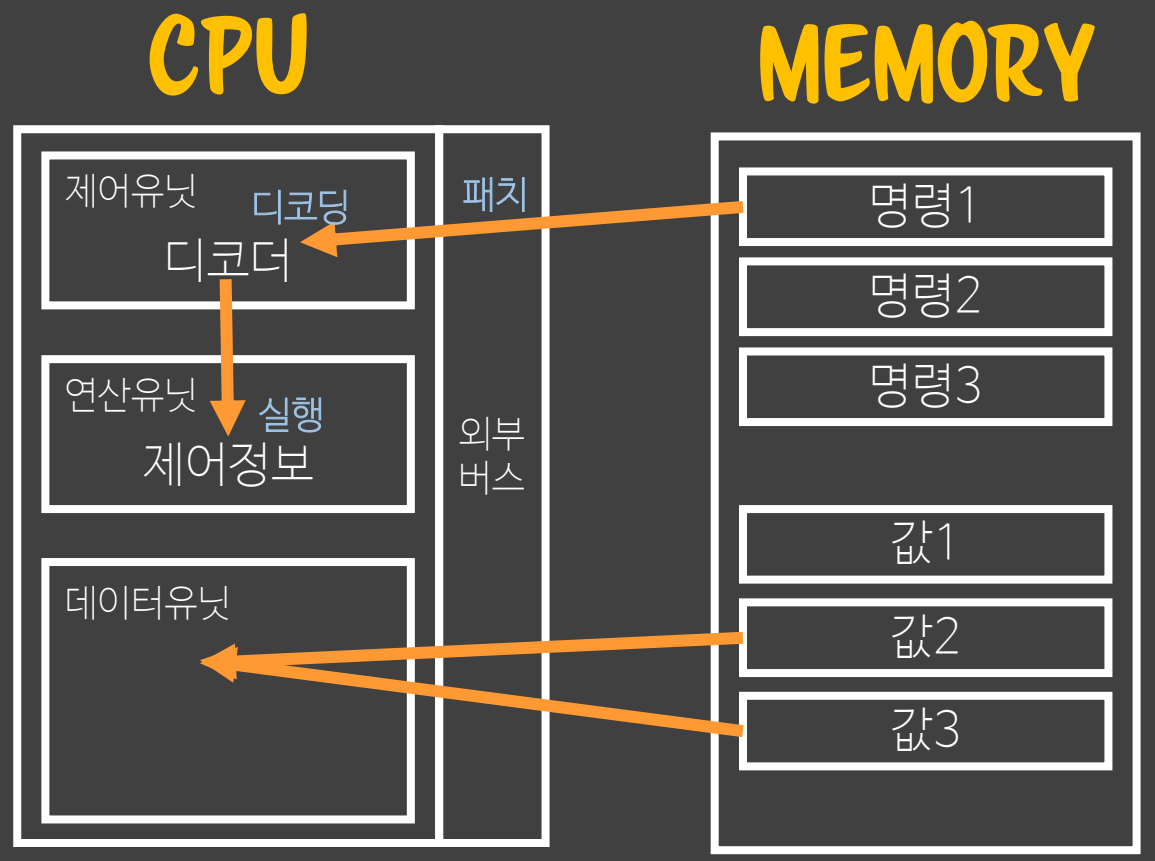
INSTRUCTION FETCH & DECODING



Runtime

LOADING

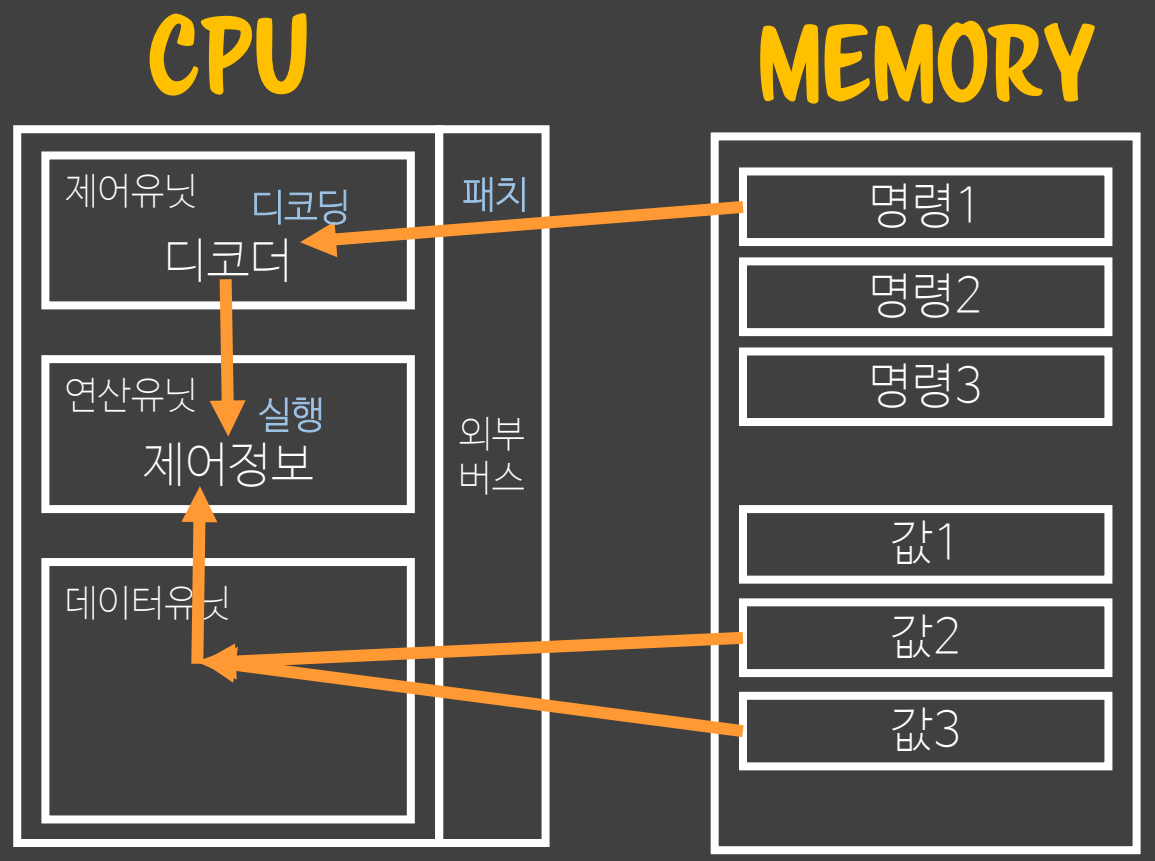
INSTRUCTION FETCH & DECODING



Runtime

LOADING

INSTRUCTION FETCH & DECODING

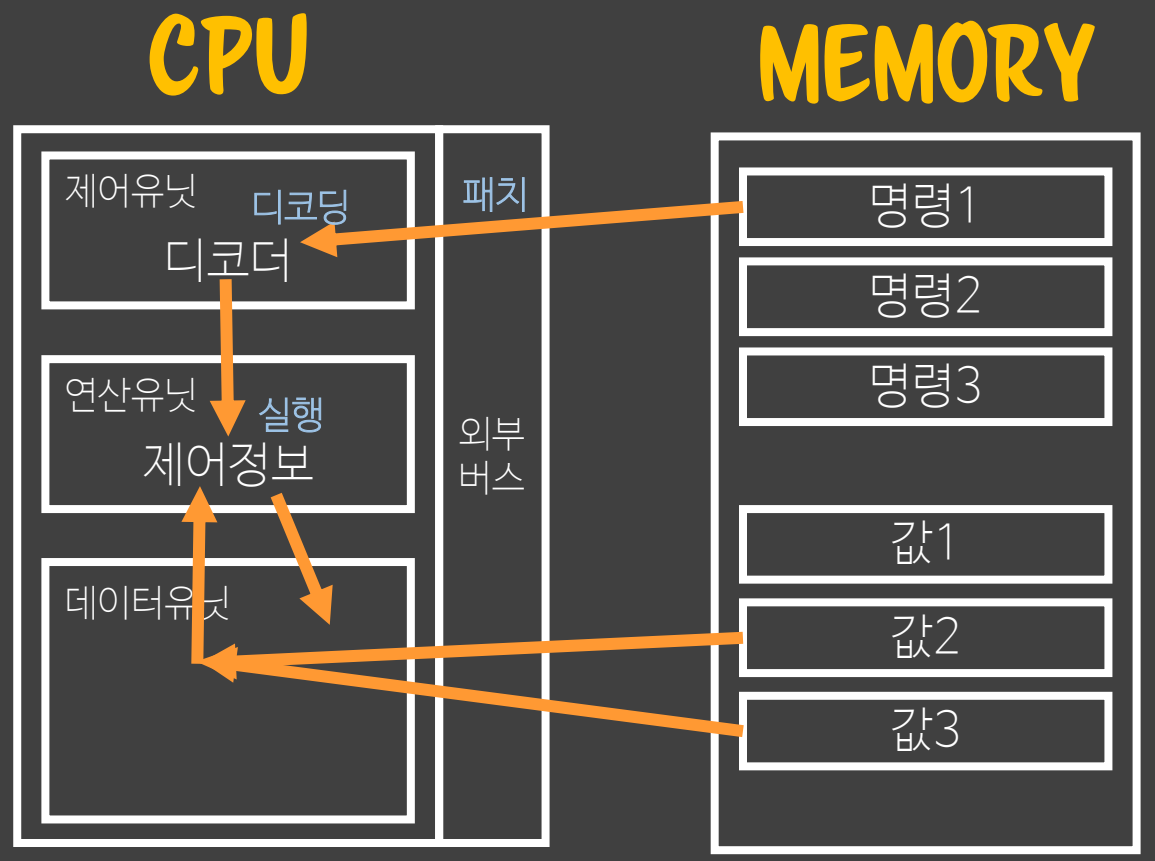


Runtime

LOADING

INSTRUCTION FETCH & DECODING

EXECUTION

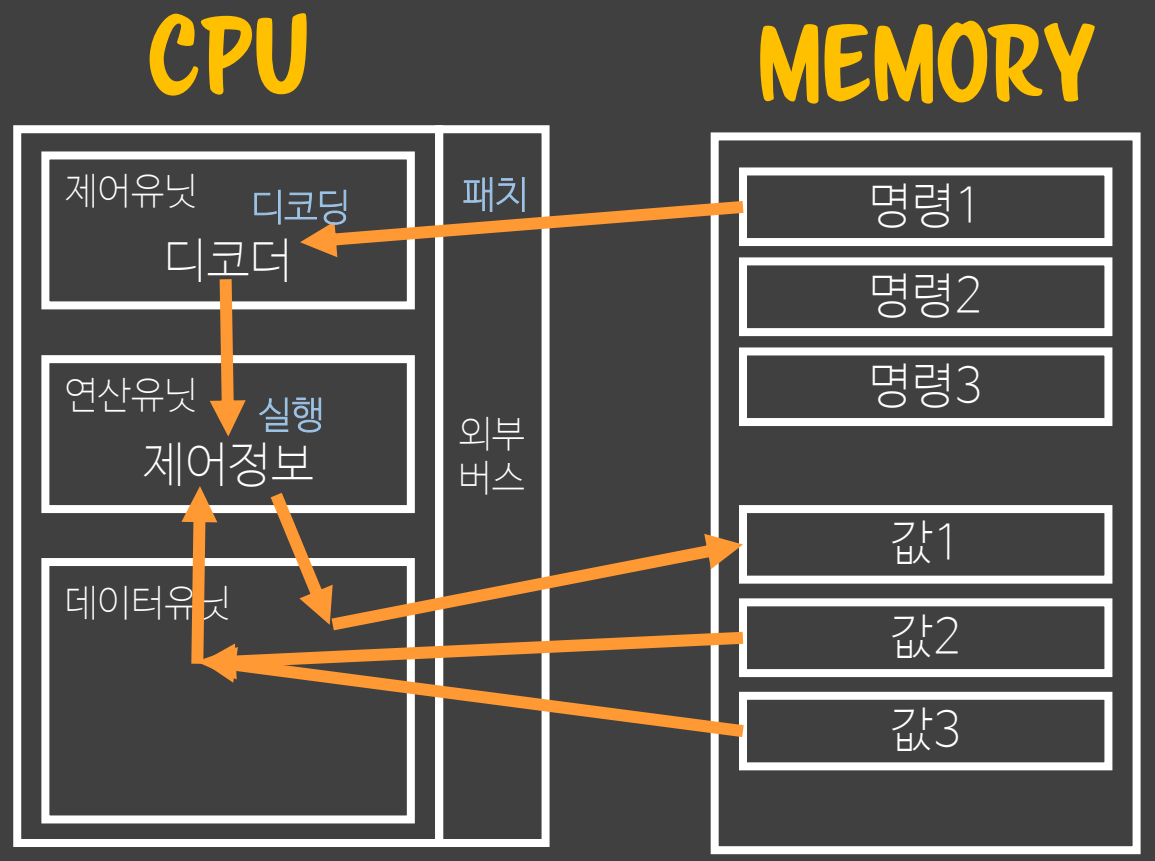


Runtime

LOADING

INSTRUCTION FETCH & DECODING

EXECUTION

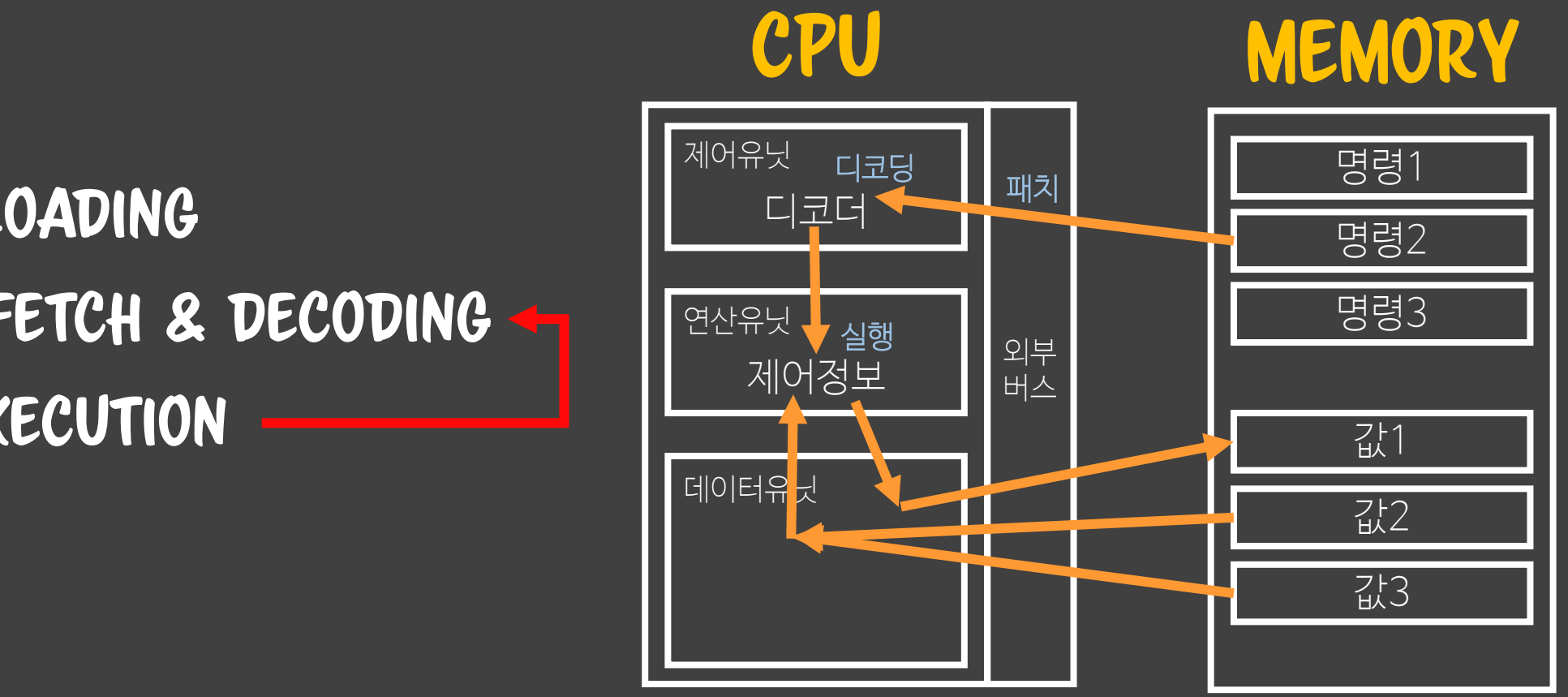


Runtime

LOADING

INSTRUCTION FETCH & DECODING

EXECUTION



Runtime

Runtime

LANGUAGE CODE — LINT TIME

MACHINE LANGUAGE — COMPILE TIME

FILE

LOAD

RUN — RUN TIME

TERMINATE

Runtime

LANGUAGE CODE — LINT TIME

MACHINE LANGUAGE — COMPILE TIME

FILE

LOAD

RUN — RUN TIME

TERMINATE

Runtime

LANGUAGE CODE — LINT TIME

MACHINE LANGUAGE — COMPILE TIME

FILE

LOAD

RUN — RUN TIME

TERMINATE

ESSENTIAL DEFINITION LOADING

Runtime

LANGUAGE CODE — LINT TIME

MACHINE LANGUAGE — COMPILE TIME

FILE

LOAD

RUN — RUN TIME

TERMINATE

ESSENTIAL DEFINITION LOADING

VTABLE MAPPING

Runtime

LANGUAGE CODE — LINT TIME

MACHINE LANGUAGE — COMPILE TIME

FILE

LOAD

RUN — RUN TIME

TERMINATE

ESSENTIAL DEFINITION LOADING

VTABLE MAPPING

RUN

Runtime

LANGUAGE CODE — LINT TIME

MACHINE LANGUAGE — COMPILE TIME

FILE

LOAD

RUN — RUN TIME

TERMINATE

ESSENTIAL DEFINITION LOADING

VTABLE MAPPING

RUN

RUNTIME DEFINITION LOADING

Runtime

LANGUAGE CODE — LINT TIME

MACHINE LANGUAGE — COMPILE TIME

FILE

LOAD

RUN — RUN TIME

TERMINATE

ESSENTIAL DEFINITION LOADING

VTABLE MAPPING

RUN

RUNTIME DEFINITION LOADING

RUN

Script Program

Script Program

LANGUAGE CODE — LINT TIME

MACHINE LANGUAGE — COMPILE TIME

FILE

LOAD

RUN — RUN TIME

TERMINATE

LANGUAGE CODE — LINT TIME

FILE

LOAD

MACHINE LANGUAGE

RUN — RUN TIME

TERMINATE

Script Program

LANGUAGE CODE — LINT TIME

FILE

LOAD

MACHINE LANGUAGE

RUN — RUN TIME

TERMINATE

Script Program

LANGUAGE CODE — LINT TIME

FILE

LOAD

MACHINE LANGUAGE

RUN — RUN TIME

TERMINATE

RUN

DECLARE BASE FUNCTION, CLASS...

Script Program

LANGUAGE CODE — LINT TIME

FILE

LOAD

MACHINE LANGUAGE

RUN — RUN TIME

TERMINATE

RUN

DECLARE BASE FUNCTION, CLASS...

DECLARE EXTENDED FUNCTION, CLASS...

Script Program

LANGUAGE CODE — LINT TIME

FILE

LOAD

MACHINE LANGUAGE

RUN — RUN TIME

TERMINATE

RUN

DECLARE BASE FUNCTION, CLASS...

STATIC TIME

RUN TIME

DECLARE EXTENDED FUNCTION, CLASS...

Script Program

LANGUAGE CODE — LINT TIME

FILE

LOAD

MACHINE LANGUAGE

RUN — RUN TIME

TERMINATE

RUN

DECLARE BASE FUNCTION, CLASS...

STATIC TIME

RUN TIME

DECLARE EXTENDED FUNCTION, CLASS...

USE FUNCTION, CLASS...

Script Program

LANGUAGE CODE — LINT TIME

FILE

LOAD

MACHINE LANGUAGE

RUN — RUN TIME

TERMINATE

RUN

DECLARE BASE FUNCTION, CLASS...

STATIC TIME

RUN TIME

DECLARE EXTENDED FUNCTION, CLASS...

STATIC TIME

RUN TIME

USE FUNCTION, CLASS...

Pointer of Pointer

00	01	02	03	04	05	06	07
08	09	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

A = "TEST"

00	01	02	03	04	05	06	07
08	09	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

A = "TEST"


00
01
02
03
04
05
06
07
08
09
10
11
T
12
E
13
S
14
T
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39

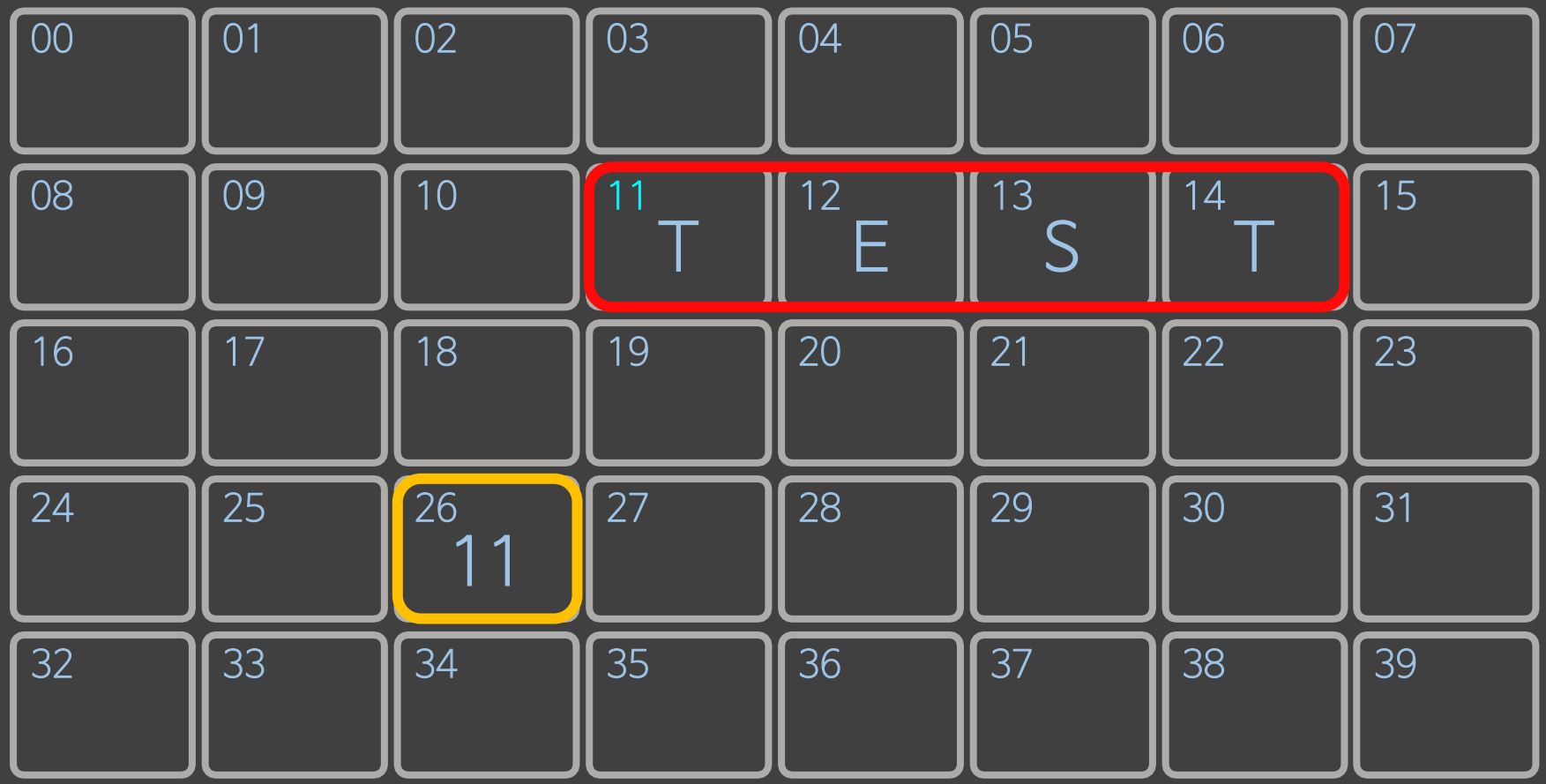
A = "TEST" &A = 11


00
01
02
03
04
05
06
07
08
09
10
11
T
12
E
13
S
14
T
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39

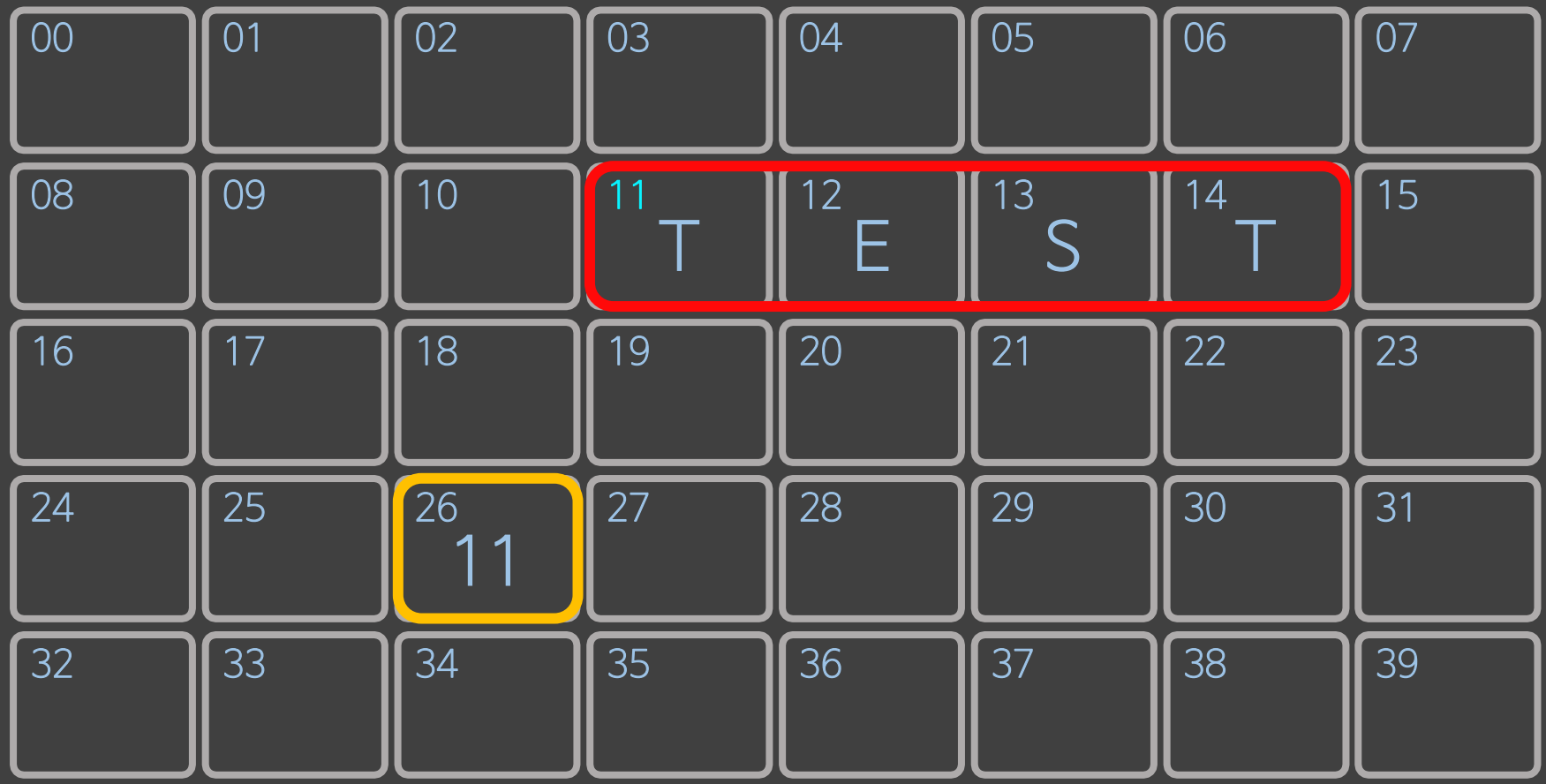
A = "TEST" &A = 11 B = &A


00
01
02
03
04
05
06
07
08
09
10
11
T
12
E
13
S
14
T
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39

A = "TEST" &A = 11 B = &A

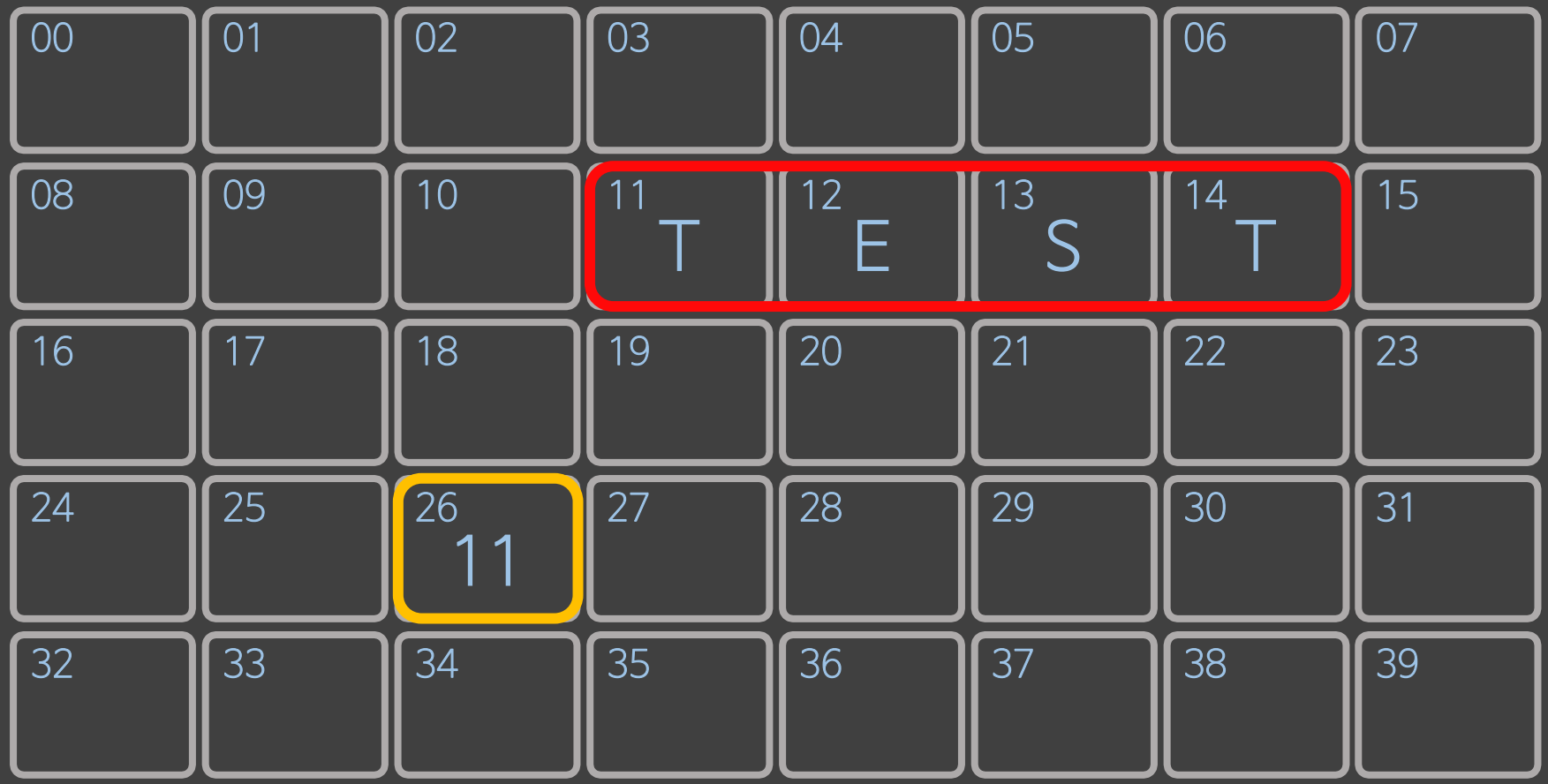

00
01
02
03
04
05
06
07
08
09
10
11
T
12
E
13
S
14
T
15
16
17
18
19
20
21
22
23
24
25
26
11
27
28
29
30
31
32
33
34
35
36
37
38
39

A = "TEST" &A = 11 B = &A *B = "TEST"



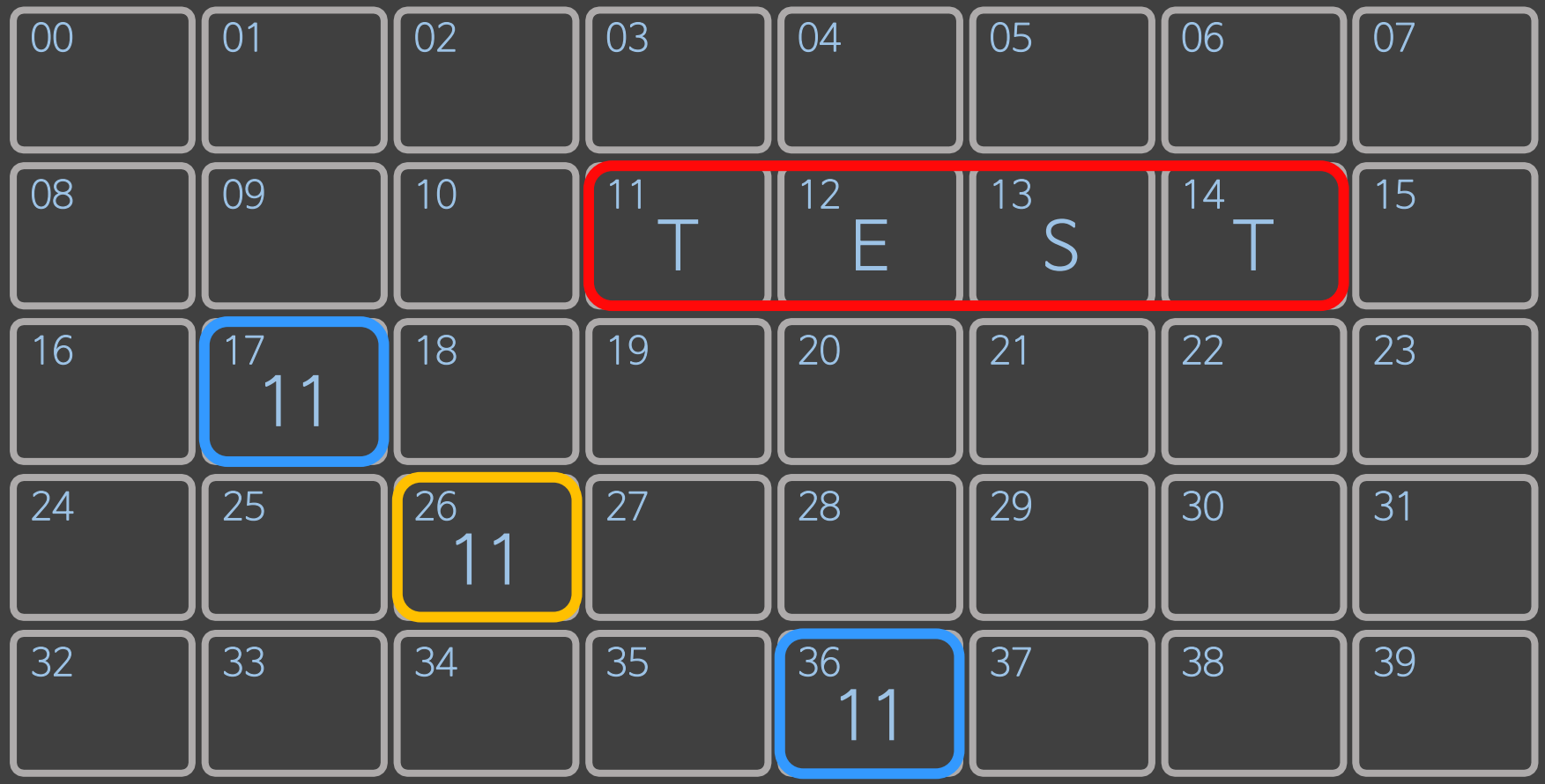
A = "TEST" &A = 11 B = &A *B = "TEST"

C = B, D = B, ...



A = "TEST" &A = 11 B = &A *B = "TEST"

C = B, D = B, ...



A = "TEST" &A = 11

B = &A *B = "TEST"

C = B, D = B, ...

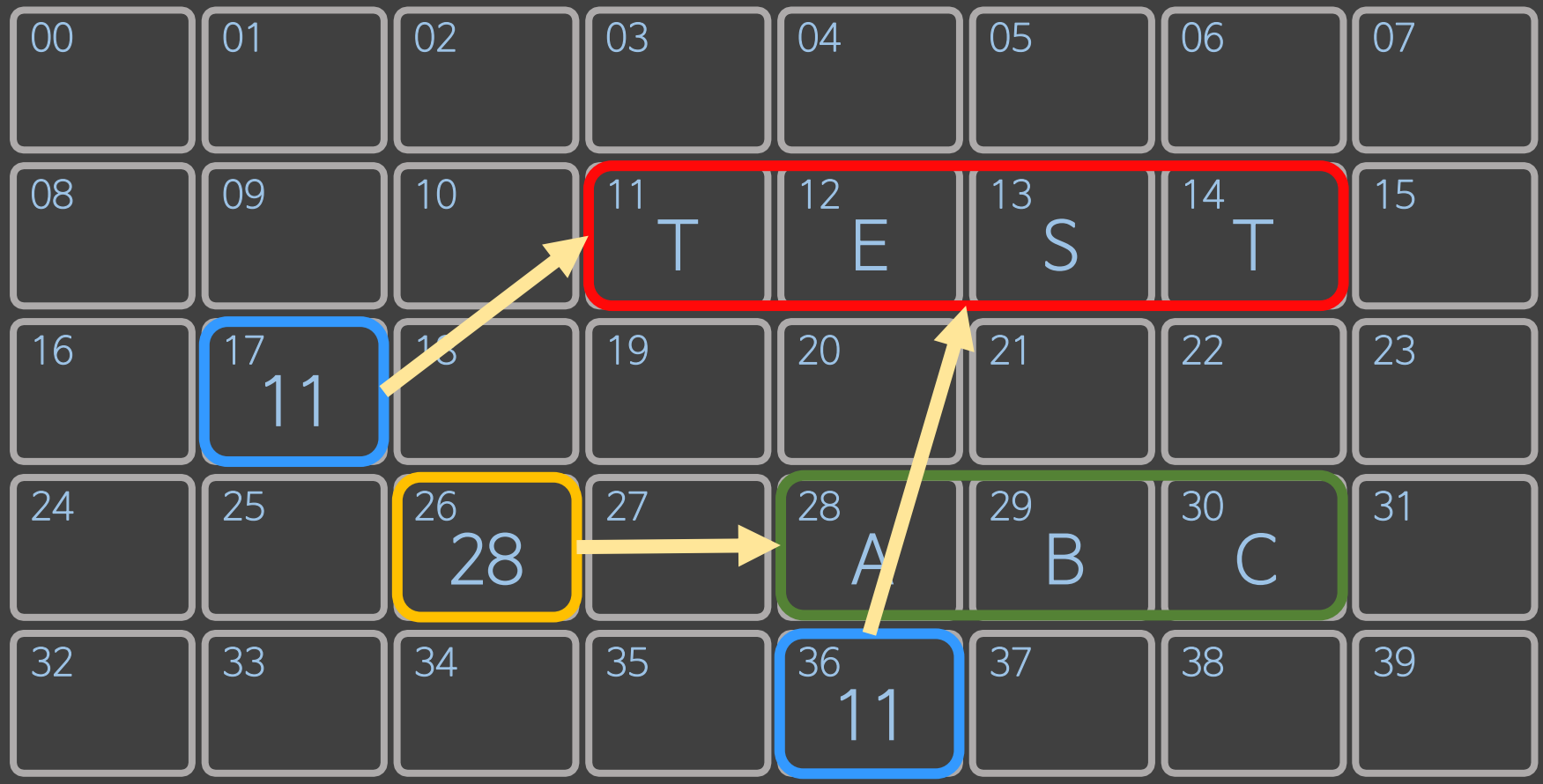
B = &K

A = "TEST" &A = 11

B = &A *B = "TEST"

C = B, D = B, ...

B = &K

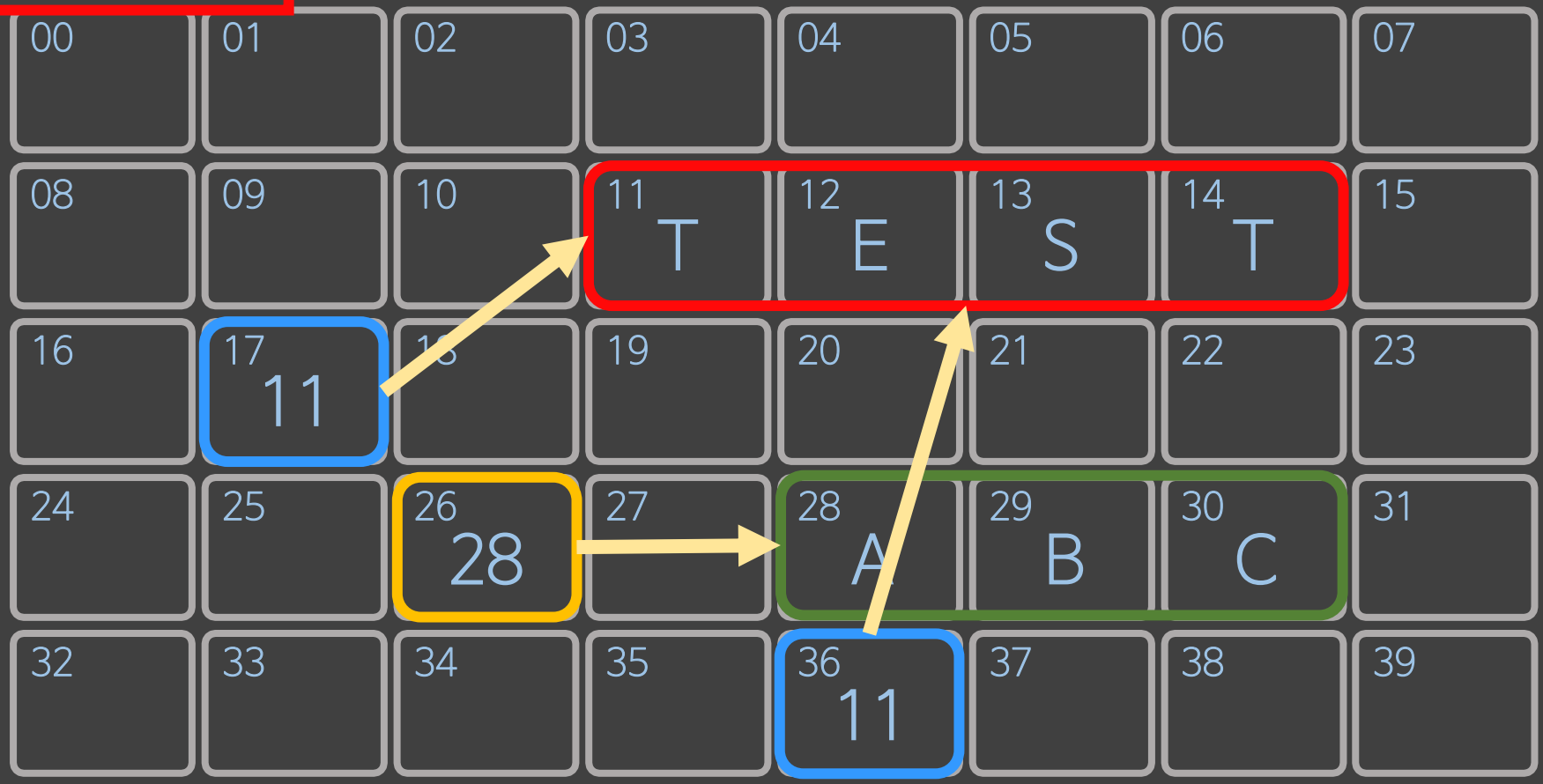


A = "TEST" &A = 11

B = &A *B = "TEST"

C = B, D = B, ...

B = &K



B = {VALUE:&A, V:3}

C = B, D = B, ...

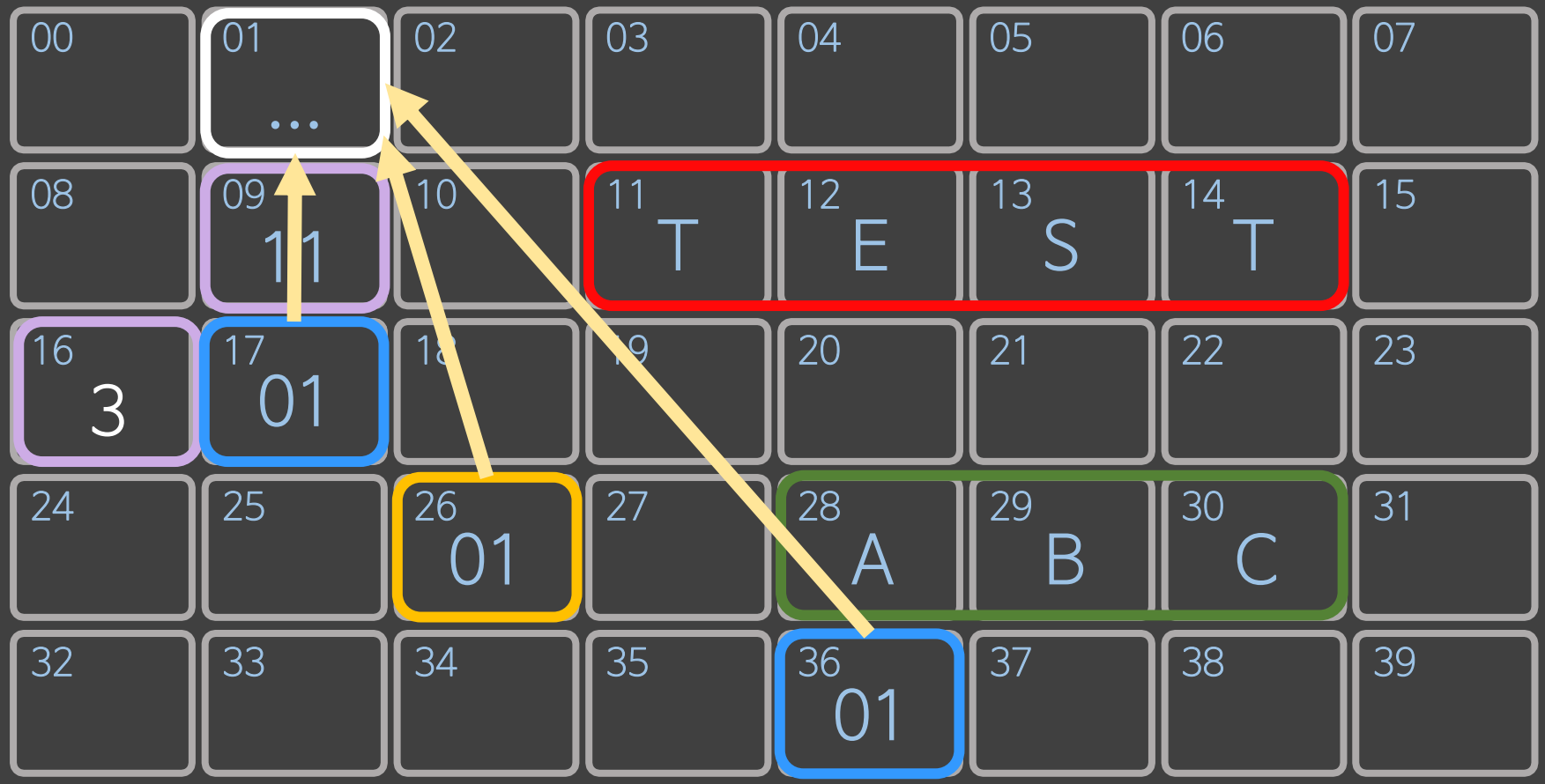
B = {VALUE:&A, V:3}

C = B, D = B, ...

B.VALUE = &K

B = {VALUE:&A, V:3}

C = B, D = B, ...



B = {VALUE:&A, V:3}

C = B, D = B, ...

B.VALUE = &K

OOP base system

Value & Identifier

```
class ValueType(val name:String){
    override operator fun equals(n:Any?) = n == name
}
```

Value & Identifier

```
class ValueType(val name:String){
    override operator fun equals(n:Any?) = n == name
}
```

```
ValueType("abc") == ValueType("abc") //true
```

Value & Identifier

```
class ValueType(val name:String){
    override operator fun equals(n:Any?) = n == name
}
```

```
ValueType("abc") == ValueType("abc") //true
```

```
ValueType("abc") === ValueType("abc") //false
```

Value & Identifier

```
class ValueType(val name:String){
    override operator fun equals(n:Any?) = n == name
}
```

```
ValueType("abc") == ValueType("abc") //true
```

```
ValueType("abc") === ValueType("abc") //false
```

Polymorphism

Polymorphism

Substituion : 대체가능성
Internal identity : 내적동질성

Polymorphism

Substituion : 대체가능성
Internal identity : 내적동질성

```
open class Worker:Runnable{
    override fun run() = println("working")
}
class HardWorker:Worker(){
    override fun run() = println("HardWorking")
}
var worker:Runnable = Worker()
println(worker.run()) // working
worker = HardWorker()
println(worker.run())
```

Polymorphism

Substituion : 대체가능성
Internal identity : 내적동질성

```
open class Worker:Runnable{
    override fun run() = println("working")
}
class HardWorker:Worker(){
    override fun run() = println("HardWorking")
}
var worker:Runnable = Worker()
println(worker.run()) // working
worker = HardWorker()
println(worker.run())
```

Polymorphism

Substituion : 대체가능성
Internal identity : 내적동질성

```
open class Worker:Runnable{
    override fun run() = println("working")
}
class HardWorker:Worker(){
    override fun run() = println("HardWorking")
}
var worker:Runnable = Worker()
println(worker.run())
worker = HardWorker()
println(worker.run()) // HardWorking
```

Polymorphism

Substituion : 대체가능성
Internal identity : 내적동질성

```
open class Worker:Runnable{
    override fun run() = println("working")
    fun print() = println(run())
}
class HardWorker:Worker(){
    override fun run() = println("HardWorking")
}
var worker:Worker = HardWorker()
println(worker.print())
```

Object

Object

Encapsulation of Functionality

Object

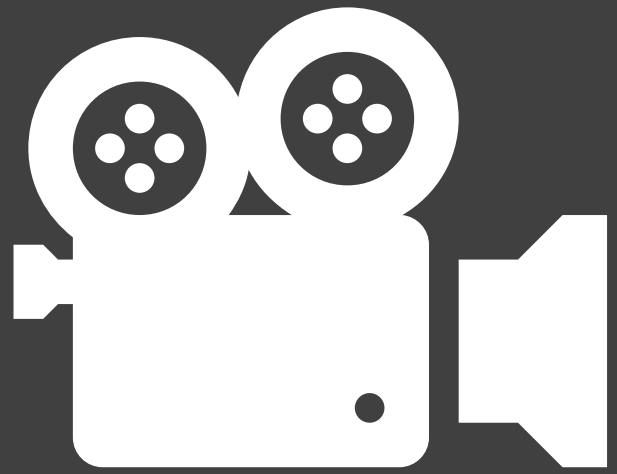
Encapsulation of Functionality
Maintenance of State

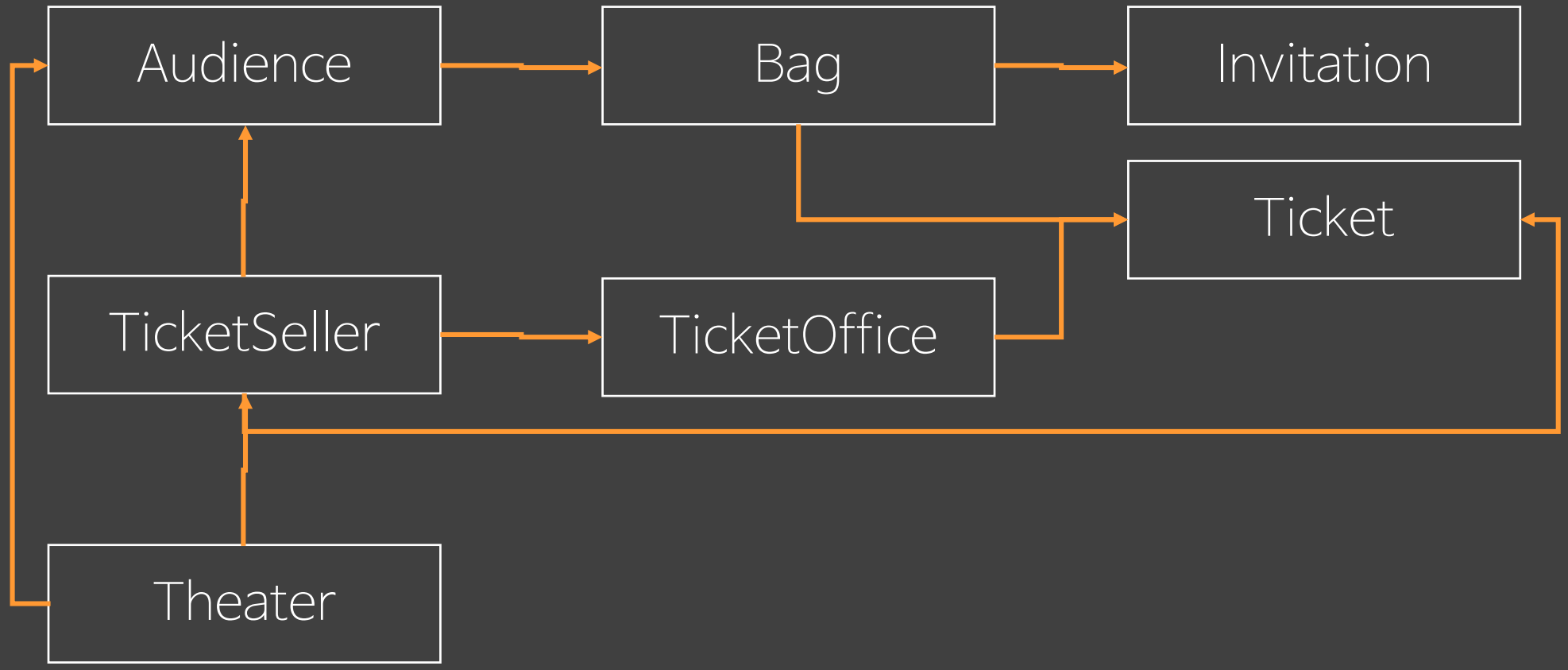
Object

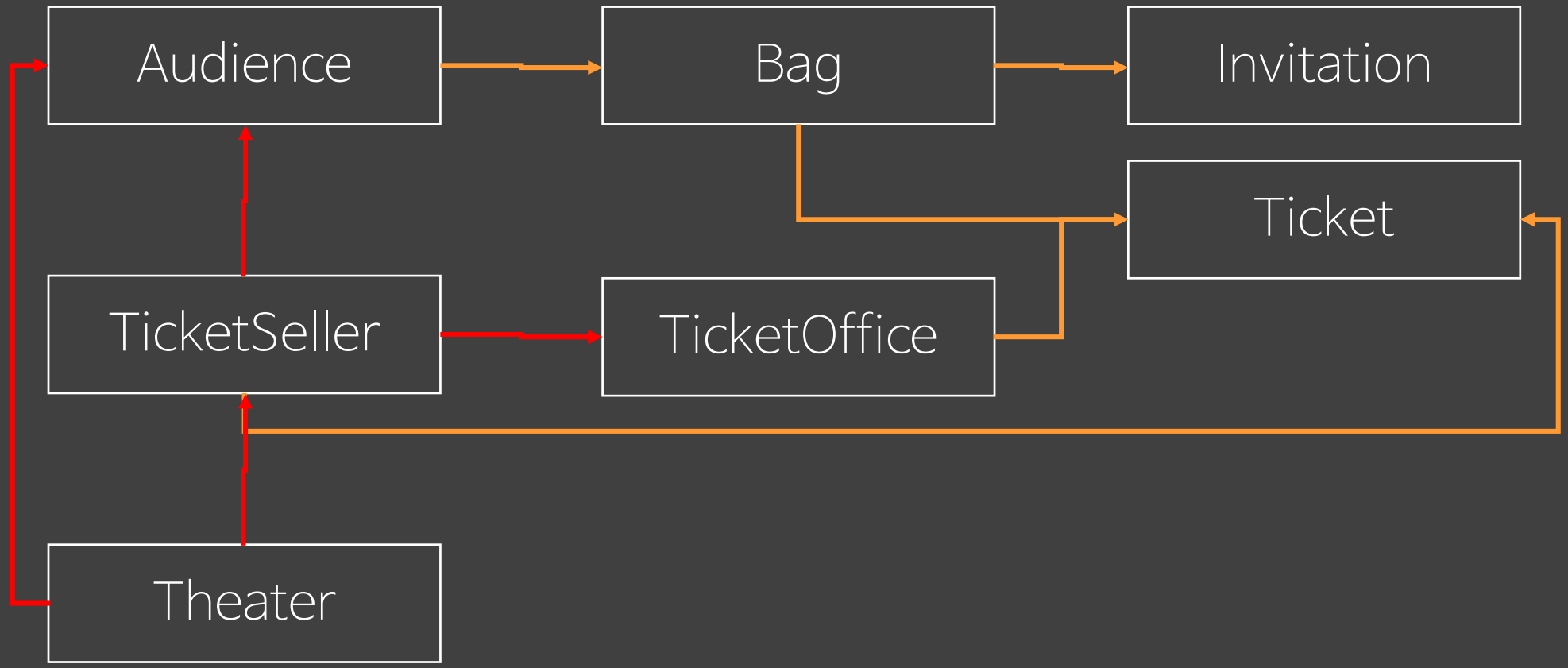
Encapsulation of Functionality
Maintenance of State

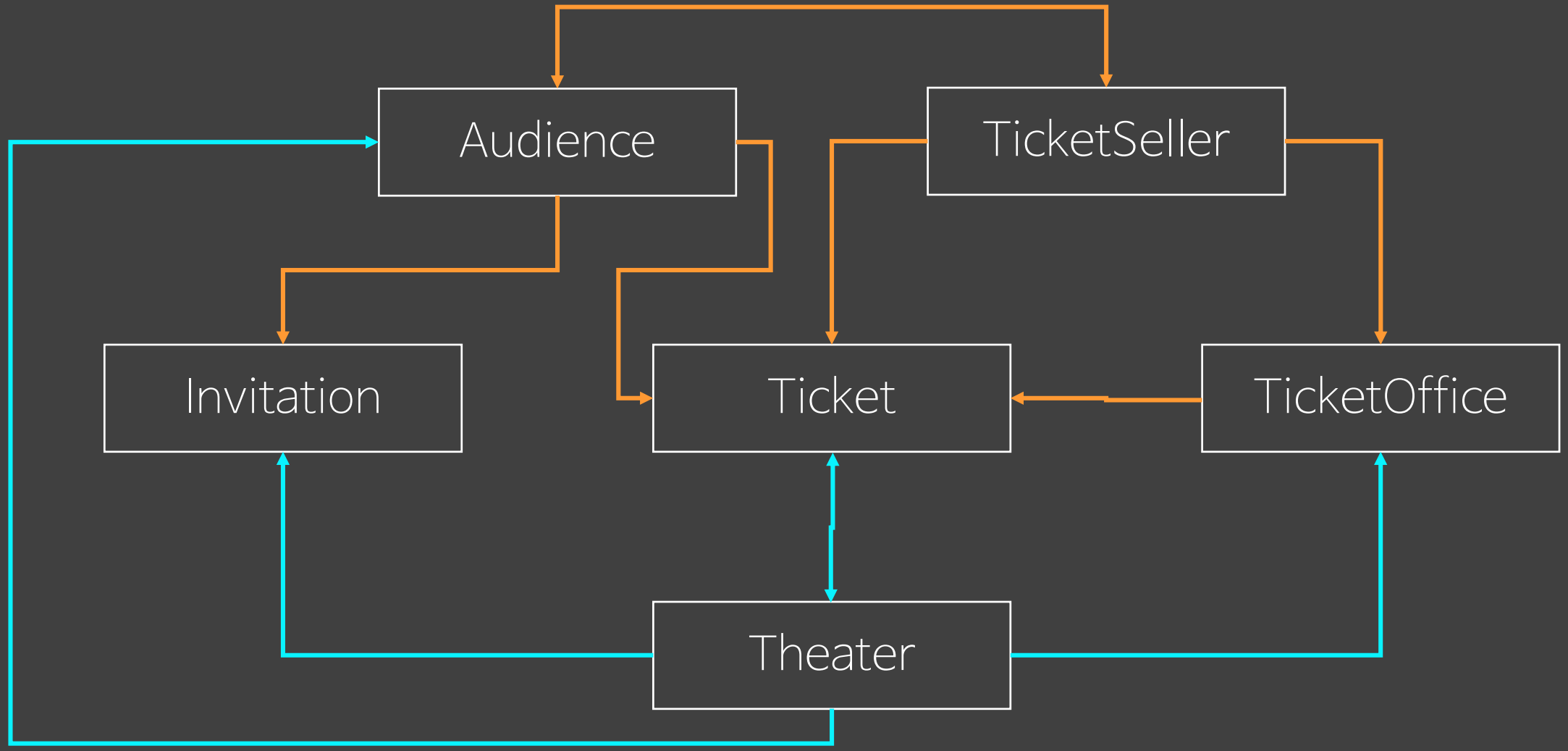
Isolation

Theater

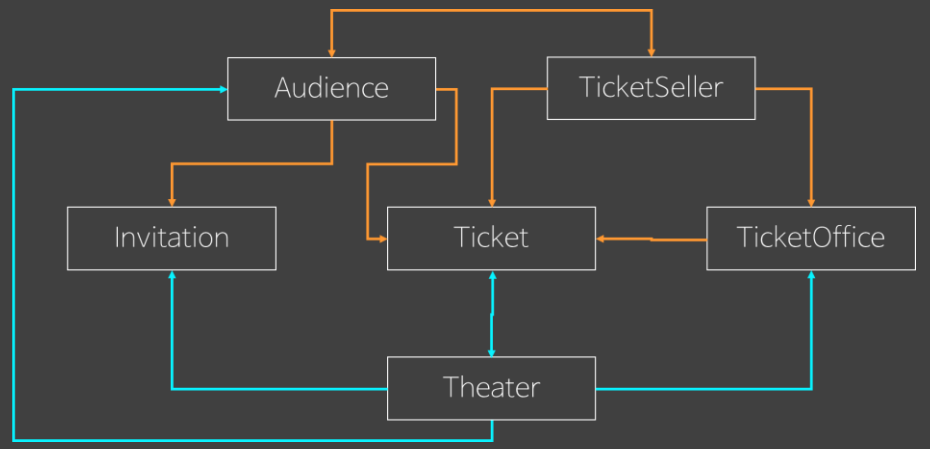



Audience
Bag
Invitation
Ticket
TicketSeller
TicketOffice
Theater

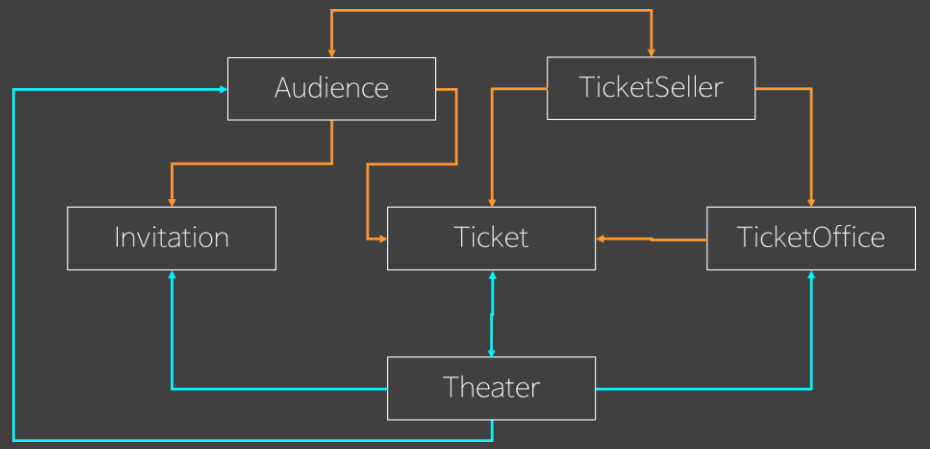

Audience
Bag
Invitation
Ticket
TicketSeller
TicketOffice
Theater


Audience
TicketSeller
Invitation
Ticket
TicketOffice
Theater

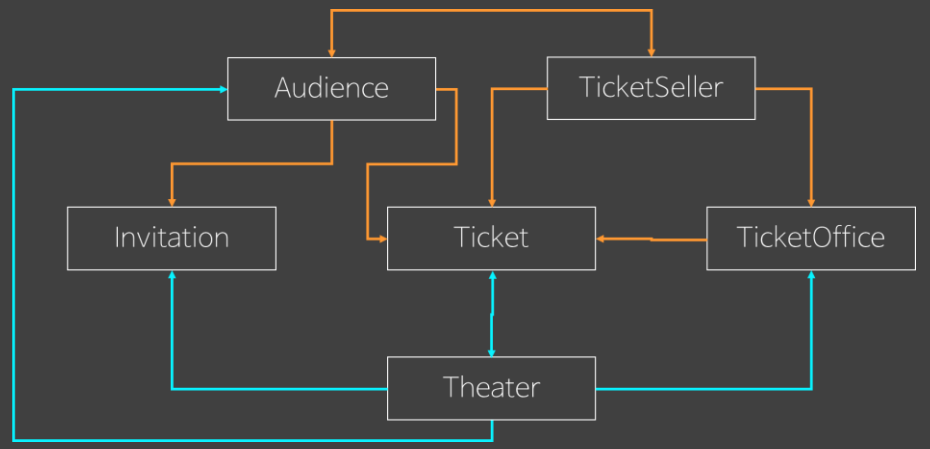
```
class Theater{
    final private List<TicketOffice> ticketOffices = new ArrayList<>();
    final private Long fee;
    public Theater(Long fee){
        this.fee = fee;
    }
    Long getFee(){
        return this.fee;
    }
    public void setTicketOffices(TicketOffice ... ticketOffices) {
        this.ticketOffices.addAll(Arrays.asList(ticketOffices));
    }
    public void setTicket(TicketOffice ticketOffice, Long num){
        if(!ticketOffices.contains(ticketOffice)) return;
        while(num-- > 0) {
            ticketOffice.addTicket(new Ticket(this));
        }
    }
    public void setInvitation(Audience audience){
        audience.setInvitation(new Invitation(this));
    }
    public boolean enter(Audience audience){
        Ticket ticket = audience.getTicket();
        return ticket.isValid(this);
    }
}
```



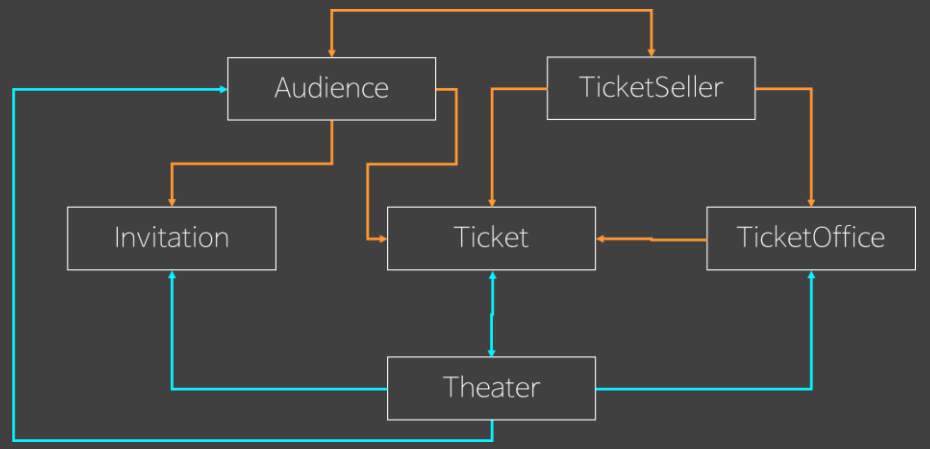
```java
class Theater{
    final private List<TicketOffice> ticketOffices = new ArrayList<>();
    final private Long fee;
    public Theater(Long fee){
        this.fee = fee;
    }
    Long getFee(){
        return this.fee;
    }
    public void setTicketOffices(TicketOffice ... ticketOffices) {
        this.ticketOffices.addAll(Arrays.asList(ticketOffices));
    }
    public void setTicket(TicketOffice ticketOffice, Long num){
        if(!ticketOffices.contains(ticketOffice)) return;
        while(num-- > 0) {
            ticketOffice.addTicket(new Ticket(this));
        }
    }
    public void setInvitation(Audience audience){
        audience.setInvitation(new Invitation(this));
    }
    public boolean enter(Audience audience){
        Ticket ticket = audience.getTicket();
        return ticket.isValid(this);
    }
}
```



```java
class Theater{
    final private List<TicketOffice> ticketOffices = new ArrayList<>();
    final private Long fee;
    public Theater(Long fee){
        this.fee = fee;
    }
    Long getFee(){
        return this.fee;
    }
    public void setTicketOffices(TicketOffice ... ticketOffices) {
        this.ticketOffices.addAll(Arrays.asList(ticketOffices));
    }
    public void setTicket(TicketOffice ticketOffice, Long num){
        if(!ticketOffices.contains(ticketOffice)) return;
        while(num-- > 0) {
            ticketOffice.addTicket(new Ticket(this));
        }
    }
    public void setInvitation(Audience audience){
        audience.setInvitation(new Invitation(this));
    }
    public boolean enter(Audience audience){
        Ticket ticket = audience.getTicket();
        return ticket.isValid(this);
    }
}
```

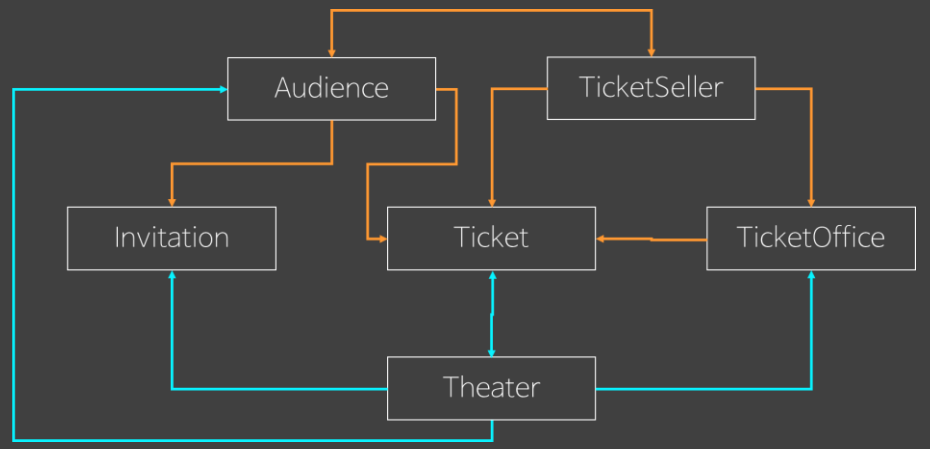


```java
class Theater{
    final private List<TicketOffice> ticketOffices = new ArrayList<>();
    final private Long fee;
    public Theater(Long fee){
        this.fee = fee;
    }
    Long getFee(){
        return this.fee;
    }
    public void setTicketOffices(TicketOffice ... ticketOffices) {
        this.ticketOffices.addAll(Arrays.asList(ticketOffices));
    }
    public void setTicket(TicketOffice ticketOffice, Long num){
        if(!ticketOffices.contains(ticketOffice)) return;
        while(num-- > 0) {
            ticketOffice.addTicket(new Ticket(this));
        }
    }
    public void setInvitation(Audience audience){
        audience.setInvitation(new Invitation(this));
    }
    public boolean enter(Audience audience){
        Ticket ticket = audience.getTicket();
        return ticket.isValid(this);
    }
}
```



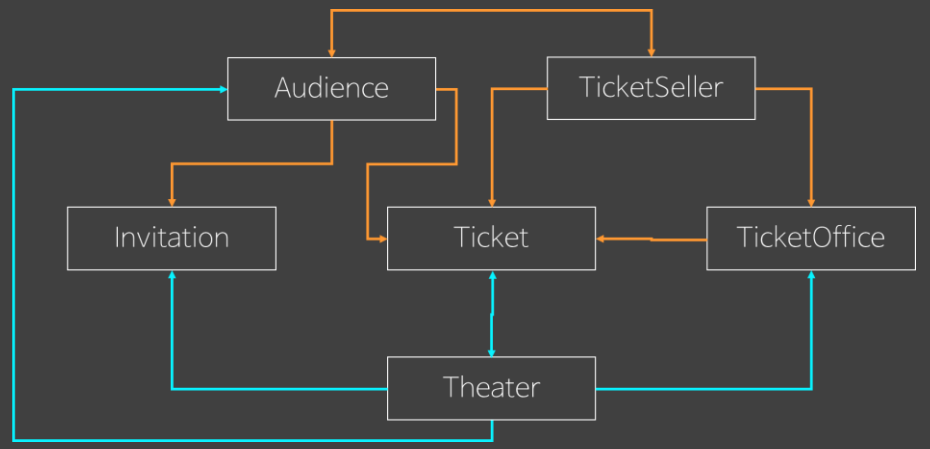
```java
public class Ticket {
    final static public Ticket EMPTY = new Ticket(null);
    final private Theater theater;
    private boolean isEntered = false;
    public Ticket(Theater theater){
        this.theater = theater;
    }
    public boolean isValid(Theater theater){
        if(isEntered || theater != this.theater || this == EMPTY){
            return false;
        }else{
            isEntered = true;
            return true;
        }
    }
    public Long getFee(){
        return theater.getFee();
    }
}
```

```java
public class Invitation {
    final static public Invitation EMPTY = new Invitation(null);
    final private Theater theater;
    public Invitation(Theater theater){
        this.theater = theater;
    }
}
```

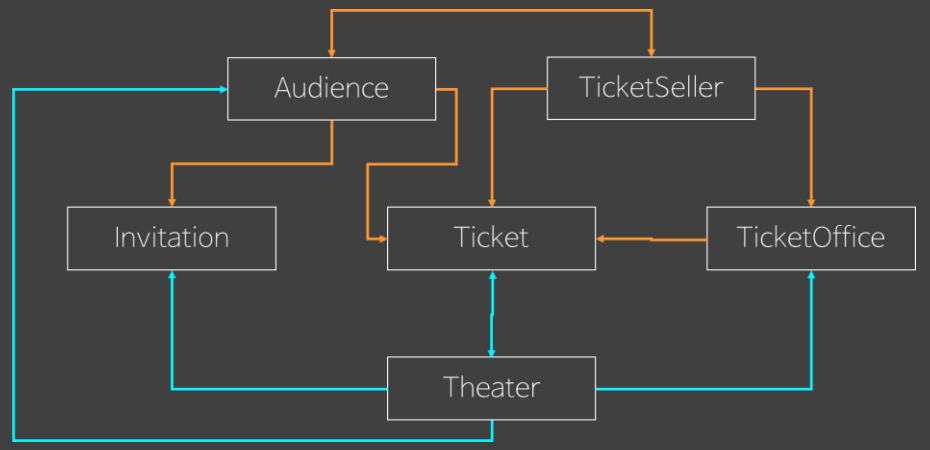


```java
public class Ticket {
    final static public Ticket EMPTY = new Ticket(null);
    final private Theater theater;
    private boolean isEntered = false;
    public Ticket(Theater theater){
        this.theater = theater;
    }
    public boolean isValid(Theater theater){
        if(isEntered || theater != this.theater || this == EMPTY){
            return false;
        }else{
            isEntered = true;
            return true;
        }
    }
    public Long getFee(){
        return theater.getFee();
    }
}
```

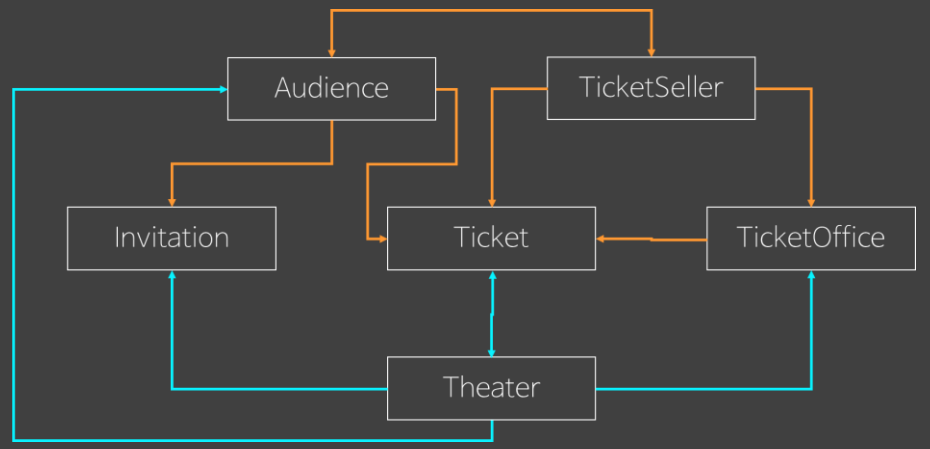
```java
public class Invitation {
    final static public Invitation EMPTY = new Invitation(null);
    final private Theater theater;
    public Invitation(Theater theater){
        this.theater = theater;
    }
}
```



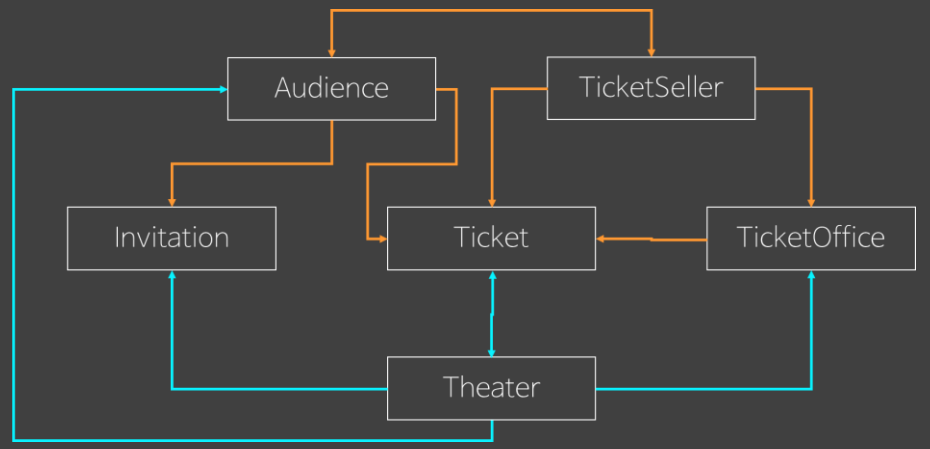
```java
public class Ticket {
    final static public Ticket EMPTY = new Ticket(null);
    final private Theater theater;
    private boolean isEntered = false;
    public Ticket(Theater theater){
        this.theater = theater;
    }
    public boolean isValid(Theater theater){
        if(isEntered || theater != this.theater || this == EMPTY){
            return false;
        }else{
            isEntered = true;
            return true;
        }
    }
    public Long getFee(){
        return theater.getFee();
    }
}

public class Invitation {
    final static public Invitation EMPTY = new Invitation(null);
    final private Theater theater;
    public Invitation(Theater theater){
        this.theater = theater;
    }
}
```



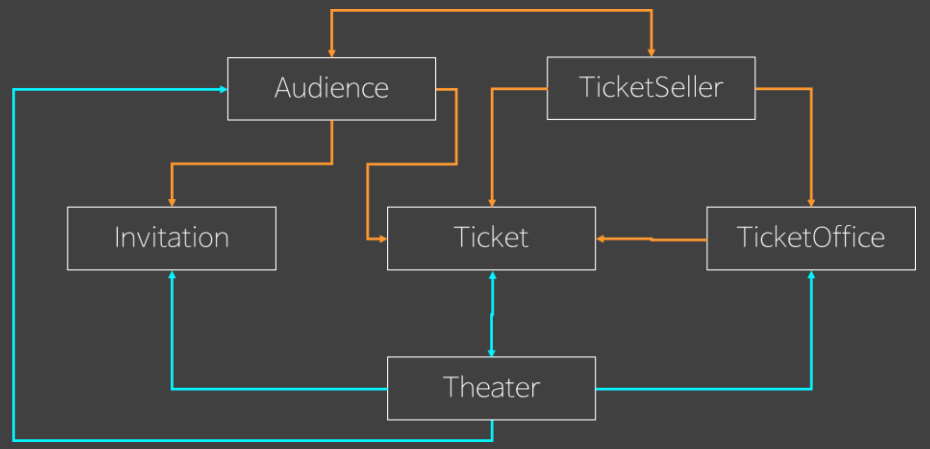
```java
public class TicketOffice {
    private Long amount;
    private List<Ticket> tickets = new ArrayList<>();
    public TicketOffice(Long amount){this.amount = amount;}
    public void addTicket(Ticket ticket){
        this.tickets.add(ticket);
    }
    public Ticket getTicketWithFee(){
        if(tickets.size() == 0) return Ticket.EMPTY;
        else{
            Ticket ticket = tickets.remove(0);
            amount += ticket.getFee();
            return ticket;
        }
    }
    public Ticket getTicketWithNoFee(){
        if(tickets.size() == 0) return Ticket.EMPTY;
        else return tickets.remove(0);
    }
    public Long getTicketPrice(){
        if(tickets.size() == 0) return 0L;
        else return tickets.get(0).getFee();
    }
}
```



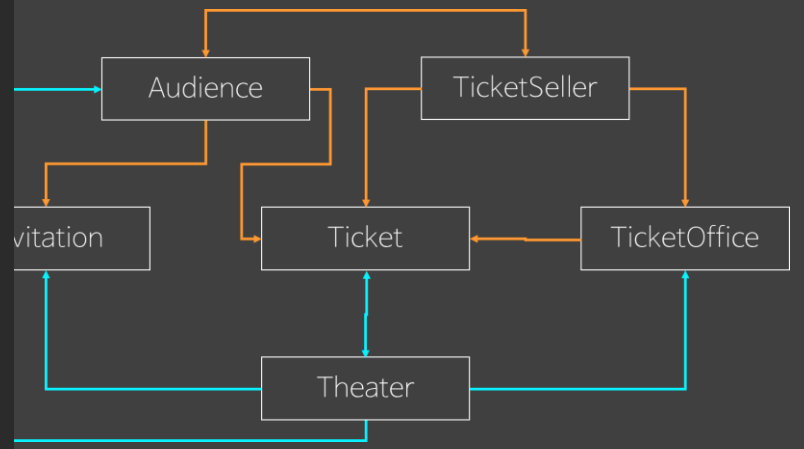
```java
public class TicketOffice {
    private Long amount;
    private List<Ticket> tickets = new ArrayList<>();
    public TicketOffice(Long amount){this.amount = amount;}
    public void addTicket(Ticket ticket){
        this.tickets.add(ticket);
    }
    public Ticket getTicketWithFee(){
        if(tickets.size() == 0) return Ticket.EMPTY;
        else{
            Ticket ticket = tickets.remove(0);
            amount += ticket.getFee();
            return ticket;
        }
    }
    public Ticket getTicketWithNoFee(){
        if(tickets.size() == 0) return Ticket.EMPTY;
        else return tickets.remove(0);
    }
    public Long getTicketPrice(){
        if(tickets.size() == 0) return 0L;
        else return tickets.get(0).getFee();
    }
}
```



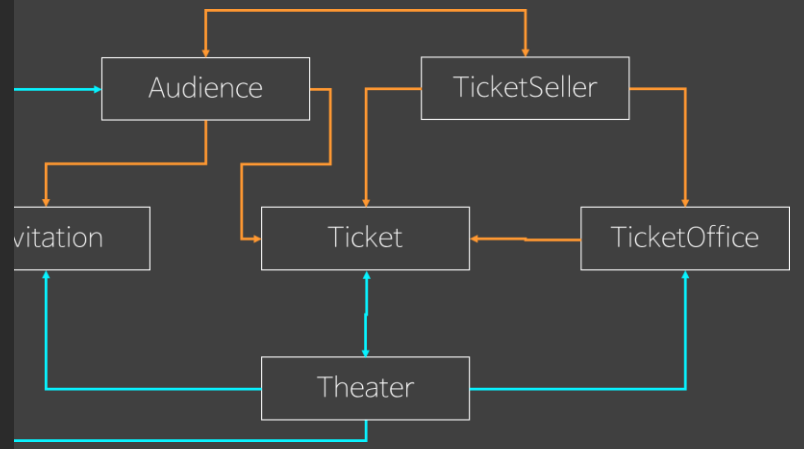
```java
public class TicketOffice {
    private Long amount;
    private List<Ticket> tickets = new ArrayList<>();
    public TicketOffice(Long amount){this.amount = amount;}
    public void addTicket(Ticket ticket){
        this.tickets.add(ticket);
    }
    public Ticket getTicketWithFee(){
        if(tickets.size() == 0) return Ticket.EMPTY;
        else{
            Ticket ticket = tickets.remove(0);
            amount += ticket.getFee();
            return ticket;
        }
    }
    public Ticket getTicketWithNoFee(){
        if(tickets.size() == 0) return Ticket.EMPTY;
        else return tickets.remove(0);
    }
    public Long getTicketPrice(){
        if(tickets.size() == 0) return 0L;
        else return tickets.get(0).getFee();
    }
}
```



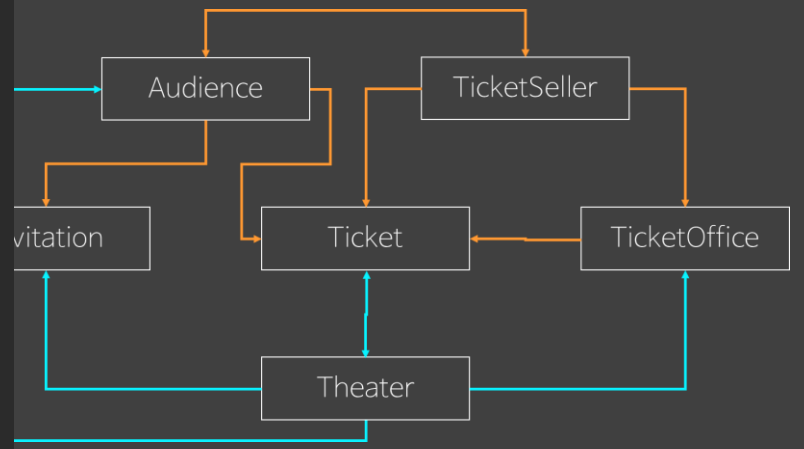
```java
public class TicketSeller {
    private TicketOffice ticketOffice;
    public void setTicketOffice(TicketOffice ticketOffice){
        this.ticketOffice = ticketOffice;
    }
    public Ticket getTicket(Audience audience){
        Ticket ticket = Ticket.EMPTY;
        if(audience.getInvitation() != Invitation.EMPTY){
            ticket = ticketOffice.getTicketWithNoFee();
            if(ticket != Ticket.EMPTY) audience.removeInvitation();
        }else{
            Long price = ticketOffice.getTicketPrice();
            if(price > 0 && audience.hasAmount(price)){
                ticket = ticketOffice.getTicketWithFee();
                if(ticket != Ticket.EMPTY) audience.minusAmount(price);
            }
        }
        return ticket;
    }
}
```



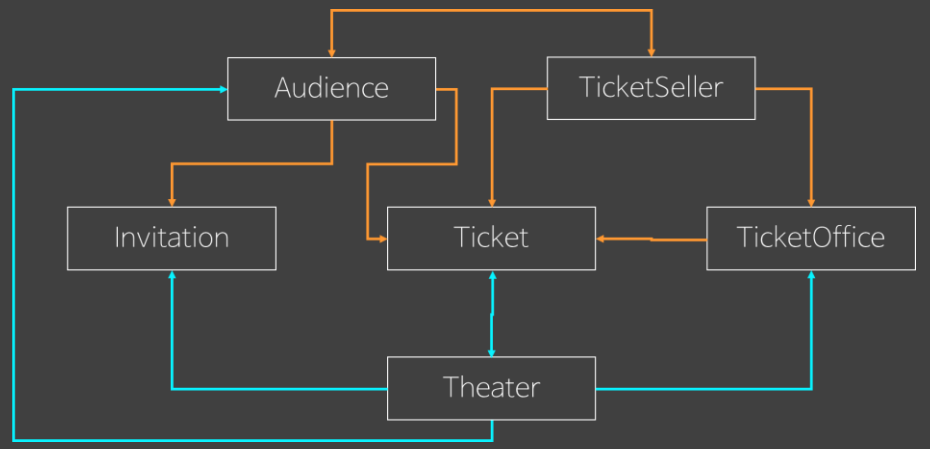
```java
public class TicketSeller {
    private TicketOffice ticketOffice;
    public void setTicketOffice(TicketOffice ticketOffice){
        this.ticketOffice = ticketOffice;
    }
    public Ticket getTicket(Audience audience){
        Ticket ticket = Ticket.EMPTY;
        if(audience.getInvitation() != Invitation.EMPTY){
            ticket = ticketOffice.getTicketWithNoFee();
            if(ticket != Ticket.EMPTY) audience.removeInvitation();
        }else{
            Long price = ticketOffice.getTicketPrice();
            if(price > 0 && audience.hasAmount(price)){
                ticket = ticketOffice.getTicketWithFee();
                if(ticket != Ticket.EMPTY) audience.minusAmount(price);
            }
        }
        return ticket;
    }
}
```



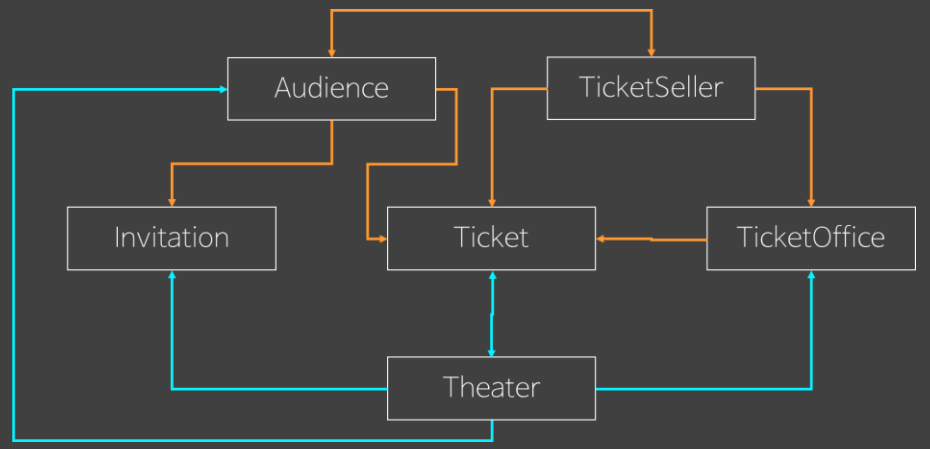
```java
public class TicketSeller {
    private TicketOffice ticketOffice;
    public void setTicketOffice(TicketOffice ticketOffice){
        this.ticketOffice = ticketOffice;
    }
    public Ticket getTicket(Audience audience){
        Ticket ticket = Ticket.EMPTY;
        if(audience.getInvitation() != Invitation.EMPTY){
            ticket = ticketOffice.getTicketWithNoFee();
            if(ticket != Ticket.EMPTY) audience.removeInvitation();
        }else{
            Long price = ticketOffice.getTicketPrice();
            if(price > 0 && audience.hasAmount(price)){
                ticket = ticketOffice.getTicketWithFee();
                if(ticket != Ticket.EMPTY) audience.minusAmount(price);
            }
        }
        return ticket;
    }
}
```



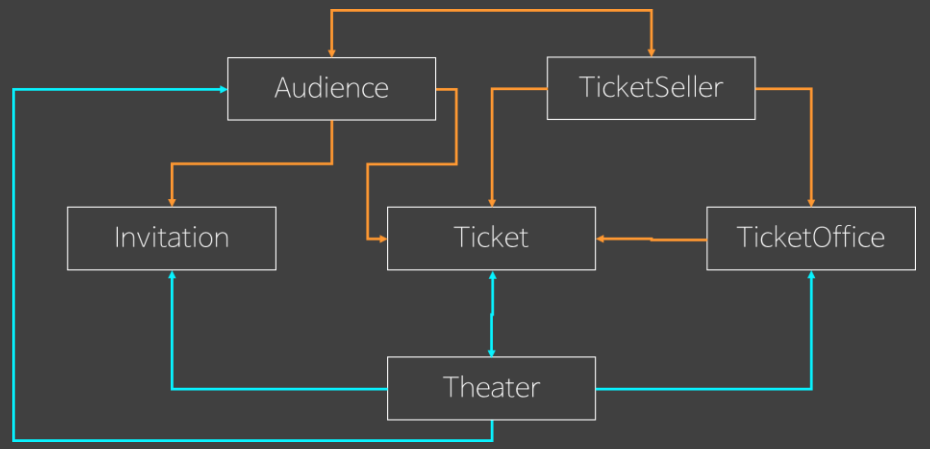
```java
class Audience {
    private Ticket ticket = Ticket.EMPTY;
    private Invitation invitation = Invitation.EMPTY;
    private Long amount;
    public Audience(Long amount){this.amount = amount;}
    public void buyTicket(TicketSeller seller){
        ticket = seller.getTicket(this);
    }
    public boolean hasAmount(Long amount){
        return this.amount >= amount;
    }
    public boolean minusAmount(Long amount){
        if(amount > this.amount) return false;
        this.amount -= amount;
        return true;
    }
    public Invitation getInvitation(){return invitation;}
    public void removeInvitation(){
        invitation = Invitation.EMPTY;
    }
    public Ticket getTicket(){return ticket;}
    public void setInvitation(Invitation invitation){
        this.invitation = invitation;
    }
}
```



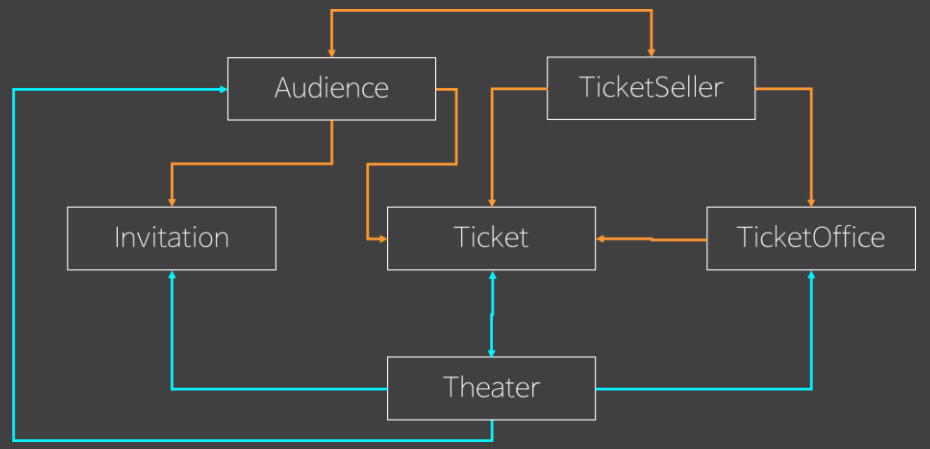
```java
class Audience {
    private Ticket ticket = Ticket.EMPTY;
    private Invitation invitation = Invitation.EMPTY;
    private Long amount;
    public Audience(Long amount){this.amount = amount;}
    public void buyTicket(TicketSeller seller){
        ticket = seller.getTicket(this);
    }
    public boolean hasAmount(Long amount){
        return this.amount >= amount;
    }
    public boolean minusAmount(Long amount){
        if(amount > this.amount) return false;
        this.amount -= amount;
        return true;
    }
    public Invitation getInvitation(){return invitation;}
    public void removeInvitation(){
        invitation = Invitation.EMPTY;
    }
    public Ticket getTicket(){return ticket;}
    public void setInvitation(Invitation invitation){
        this.invitation = invitation;
    }
}
```



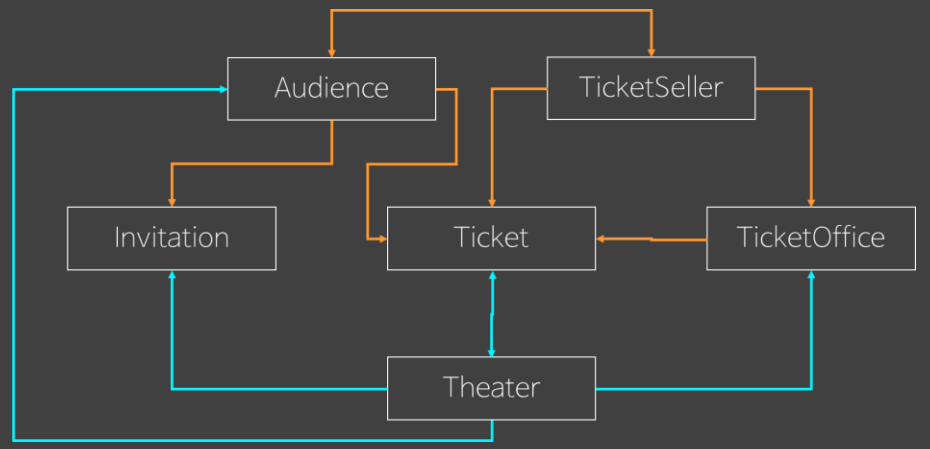
```java
class Audience {
    private Ticket ticket = Ticket.EMPTY;
    private Invitation invitation = Invitation.EMPTY;
    private Long amount;
    public Audience(Long amount){this.amount = amount;}
    public void buyTicket(TicketSeller seller){
        ticket = seller.getTicket(this);
    }
    public boolean hasAmount(Long amount){
        return this.amount >= amount;
    }
    public boolean minusAmount(Long amount){
        if(amount > this.amount) return false;
        this.amount -= amount;
        return true;
    }
    public Invitation getInvitation(){return invitation;}
    public void removeInvitation(){
        invitation = Invitation.EMPTY;
    }
    public Ticket getTicket(){return ticket;}
    public void setInvitation(Invitation invitation){
        this.invitation = invitation;
    }
}
```



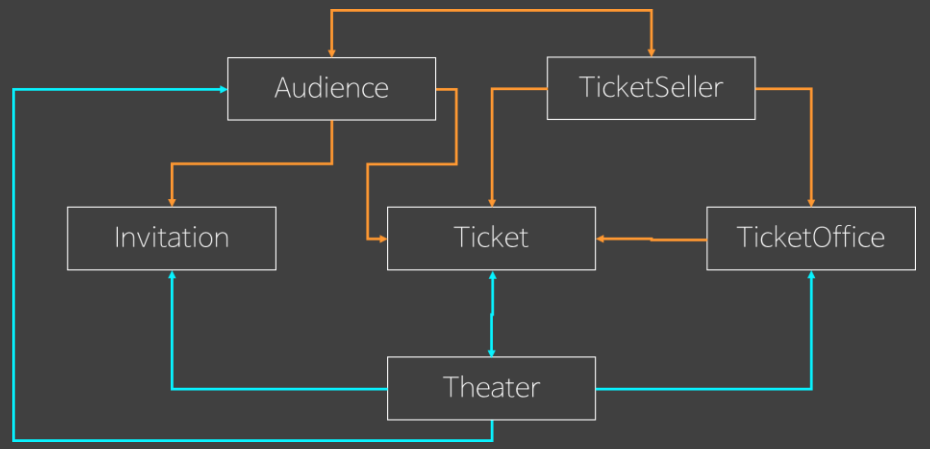
```java
public class Main {
    public static void main(String[] args) {
        Theater theater = new Theater(100L);
        Audience audience1 = new Audience(0L);
        Audience audience2 = new Audience(50L);
        TicketOffice ticketOffice = new TicketOffice(0L);
        TicketSeller seller = new TicketSeller();

        theater.setTicketOffices(ticketOffice);
        theater.setTicket(ticketOffice, 10L);
        theater.setInvitation(audience1);

        seller.setTicketOffice(ticketOffice);

        audience1.buyTicket(seller);
        audience2.buyTicket(seller);

        boolean isOk1 = theater.enter(audience1);
        boolean isOk2 = theater.enter(audience2);

        System.out.println(isOk1);
        System.out.println(isOk2);
    }
}
```



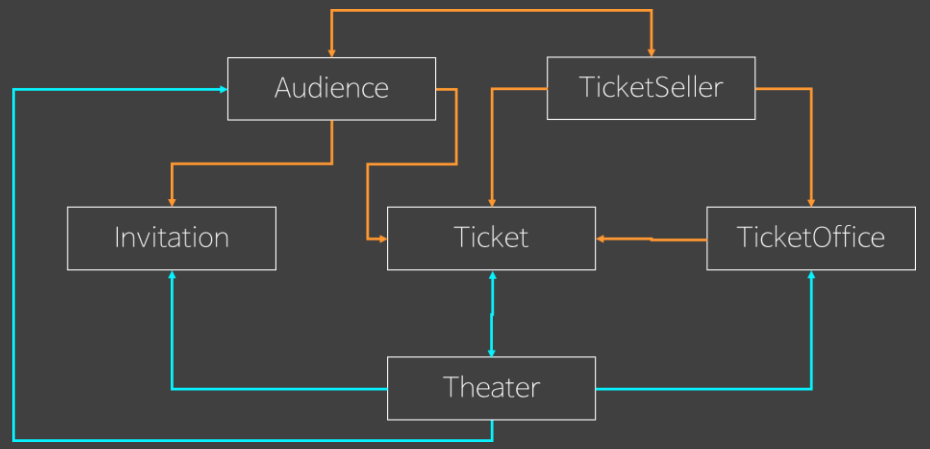
```java
public class Main {
    public static void main(String[] args) {
        Theater theater = new Theater(100L);
        Audience audience1 = new Audience(0L);
        Audience audience2 = new Audience(50L);
        TicketOffice ticketOffice = new TicketOffice(0L);
        TicketSeller seller = new TicketSeller();

        theater.setTicketOffices(ticketOffice);
        theater.setTicket(ticketOffice, 10L);
        theater.setInvitation(audience1);

        seller.setTicketOffice(ticketOffice);

        audience1.buyTicket(seller);
        audience2.buyTicket(seller);

        boolean isOk1 = theater.enter(audience1);
        boolean isOk2 = theater.enter(audience2);

        System.out.println(isOk1);
        System.out.println(isOk2);
    }
}
```



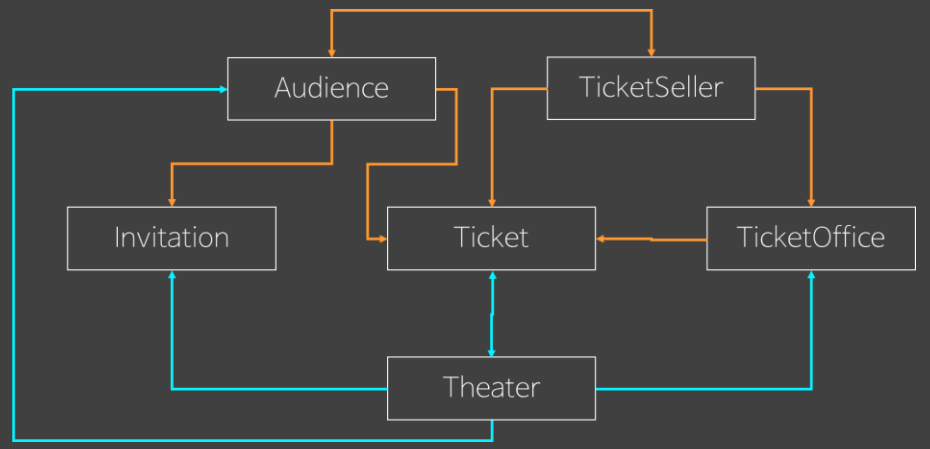
```java
public class Main {
    public static void main(String[] args) {
        Theater theater = new Theater(100L);
        Audience audience1 = new Audience(0L);
        Audience audience2 = new Audience(50L);
        TicketOffice ticketOffice = new TicketOffice(0L);
        TicketSeller seller = new TicketSeller();

        theater.setTicketOffices(ticketOffice);
        theater.setTicket(ticketOffice, 10L);
        theater.setInvitation(audience1);

        seller.setTicketOffice(ticketOffice);

        audience1.buyTicket(seller);
        audience2.buyTicket(seller);

        boolean isOk1 = theater.enter(audience1);
        boolean isOk2 = theater.enter(audience2);

        System.out.println(isOk1);
        System.out.println(isOk2);
    }
}
```



```java
public class Main {
    public static void main(String[] args) {
        Theater theater = new Theater(100L);
        Audience audience1 = new Audience(0L);
        Audience audience2 = new Audience(50L);
        TicketOffice ticketOffice = new TicketOffice(0L);
        TicketSeller seller = new TicketSeller();

        theater.setTicketOffices(ticketOffice);
        theater.setTicket(ticketOffice, 10L);
        theater.setInvitation(audience1);

        seller.setTicketOffice(ticketOffice);

        audience1.buyTicket(seller);
        audience2.buyTicket(seller);

        boolean isOk1 = theater.enter(audience1);
        boolean isOk2 = theater.enter(audience2);

        System.out.println(isOk1);
        System.out.println(isOk2);
    }
}
```



```java
public class Main {
    public static void main(String[] args) {
        Theater theater = new Theater(100L);
        Audience audience1 = new Audience(0L);
        Audience audience2 = new Audience(50L);
        TicketOffice ticketOffice = new TicketOffice(0L);
        TicketSeller seller = new TicketSeller();

        theater.setTicketOffices(ticketOffice);
        theater.setTicket(ticketOffice, 10L);
        theater.setInvitation(audience1);

        seller.setTicketOffice(ticketOffice);

        audience1.buyTicket(seller);
        audience2.buyTicket(seller);

        boolean isOk1 = theater.enter(audience1);
        boolean isOk2 = theater.enter(audience2);

        System.out.println(isOk1);
        System.out.println(isOk2);
    }
}
```



Practice

practice #1

마지막 theater예제에서
TicketOffice는 암묵적으로 하나의 극장하고만 계약하고 있다는 가정이 있게 구현되어있다.
코드 상 이 조건을 강제하도록 개선하라.

practice #2

마지막 theater예제에서
Theater는 단 하나의 영화만 고정가격으로 상영 중이다.
다양한 가격의 영화를 상영할 수 있게 개선하라.
(Movie클래스가 새롭게 필요하고 또한 이에 따라 초대,
티켓, 티켓 오피스등의 총괄적인 변화가 일어남)