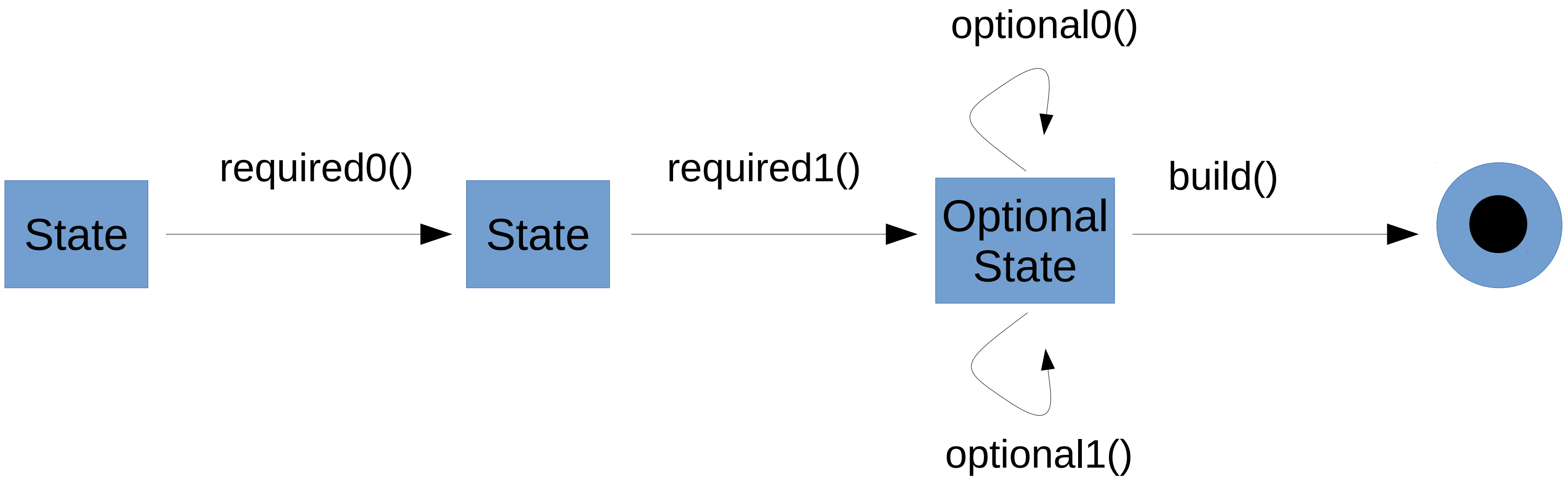
```
Ex: Intent intent = new IntentBuider(Context)
                                      .required0()
                                      .required1()
                                      .optional0()
                                      .build();
```



A intent builder is a small state machine that lets developer conviently add stuff into intent's bundles.

```
Ex: Intent intent = new IntentBuider(Context)
                        .requiredA()
                        .requiredC()
                        .optional0()
                        .build();
```

```
Ex: Intent intent = new SubIntentBuider(Context)
                        .requiredA()
                        .requiredB()
                        .requiredC()
                        .requiredD()
                        .optional0()
                        .optional1()
                        .build();
```

Our main problem is to support inheritance of IntentBuilders efficiently across modules.

Why is inheritance across modules so special ?

- it means we receive only a partial view of the information we need to create an intent builder: the super classes' annotations are not passed to the compiler, if we want to use them, it would imply to re-scan all the hierarchy of the subclass and redo a lot of work that was done.

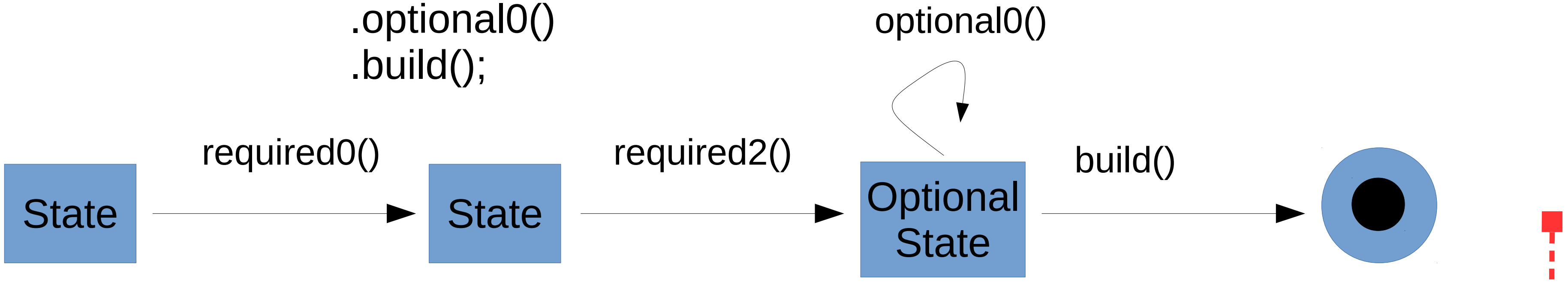
- this problem is actually the same as incremental annotation processing: during an incremental build, we will have to deal with a partial view of some classes, so classes that are relevant might not be re-compiled.

→ Achieving this efficiently is a key to fast build times.
→ The best strategy is to reuse the intent builder of the immediate super class as:

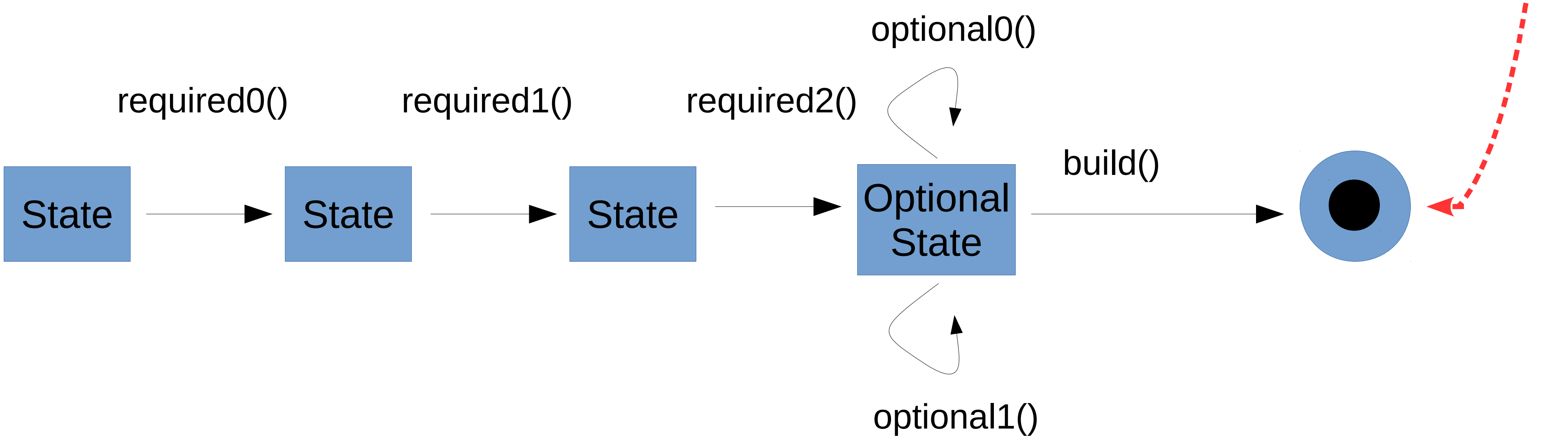
- It would not propagate to more than 1 level of hierarchy
- Some work is already done in the super class generated intent builder and if we can simply call it, we save time during the processing of sub classes.

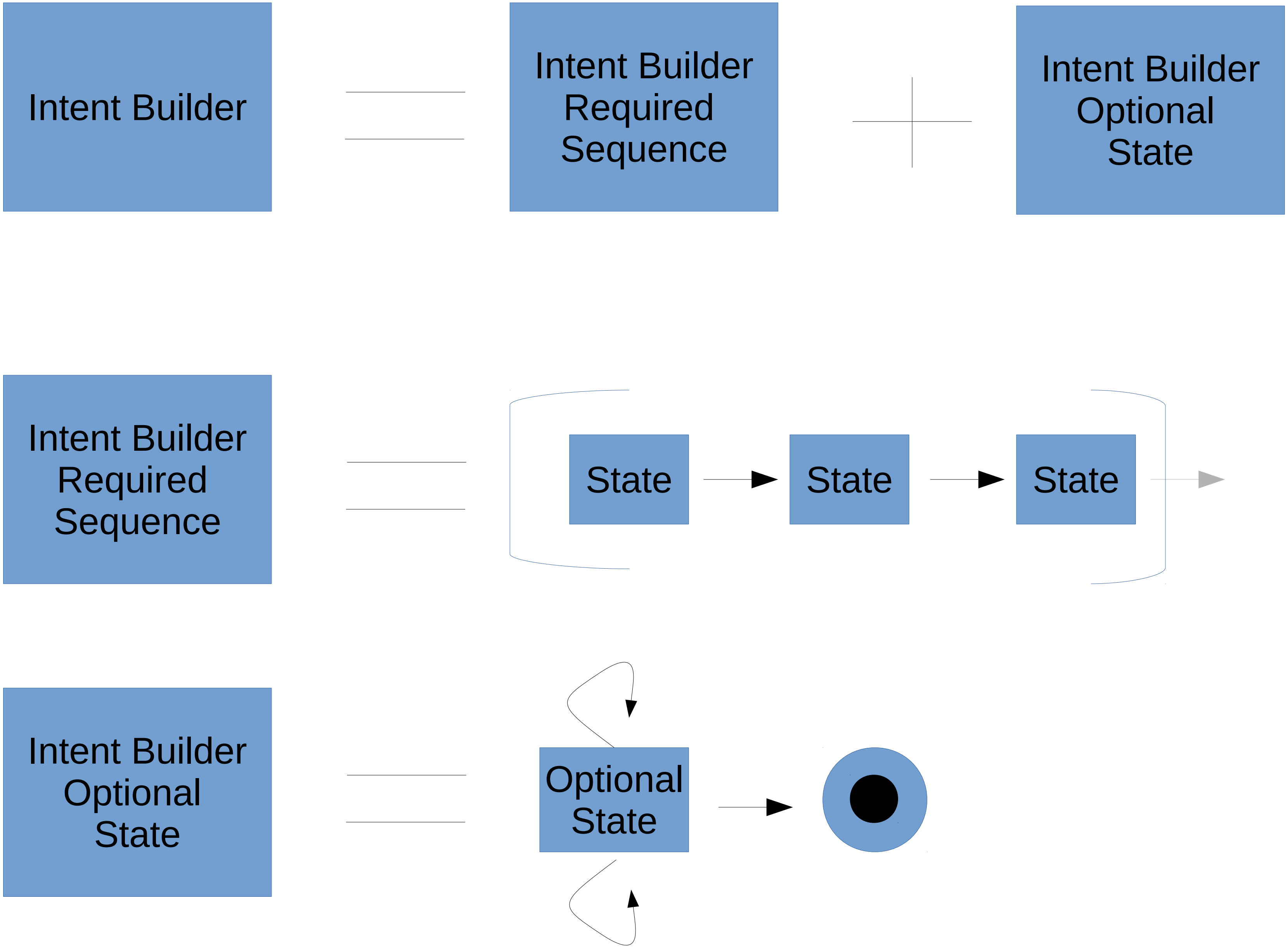
But the StateMachines of an intent builder and its super class looks so different !!! How can we achieve this ?

```
Ex: Intent intent = new IntentBuider(Context)
                        .required0()
                        .required2()
                        .optional0()
                        .build();
```



```
Ex: Intent intent = new SubIntentBuider(Context)
                        .required0()
                        .required1()
                        .required2()
                        .optional0()
                        .build();
```

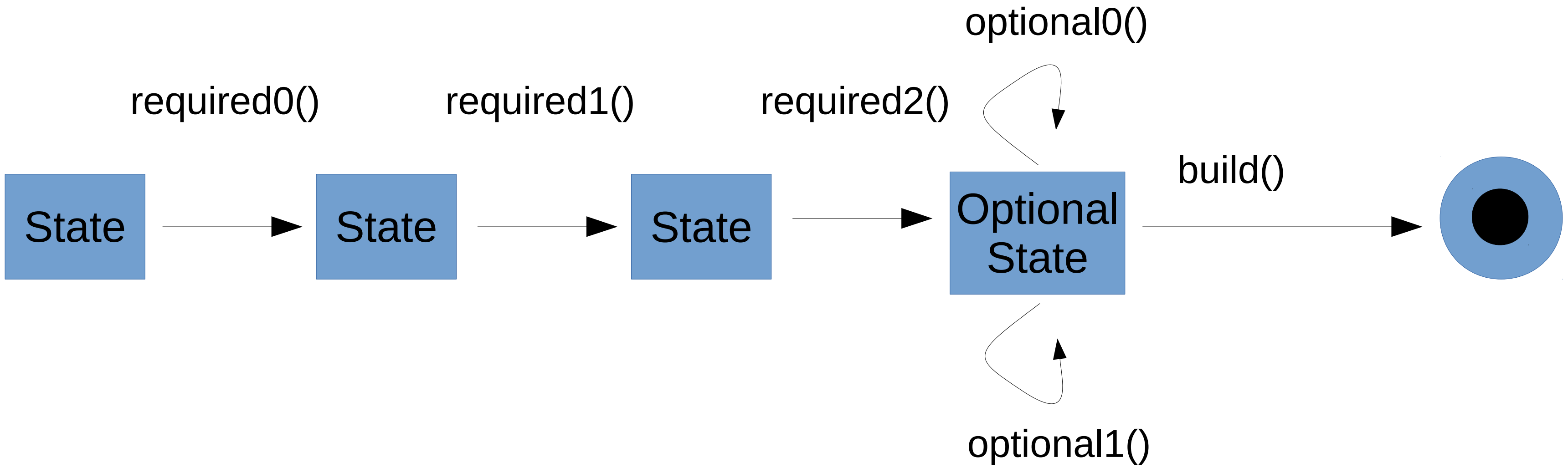


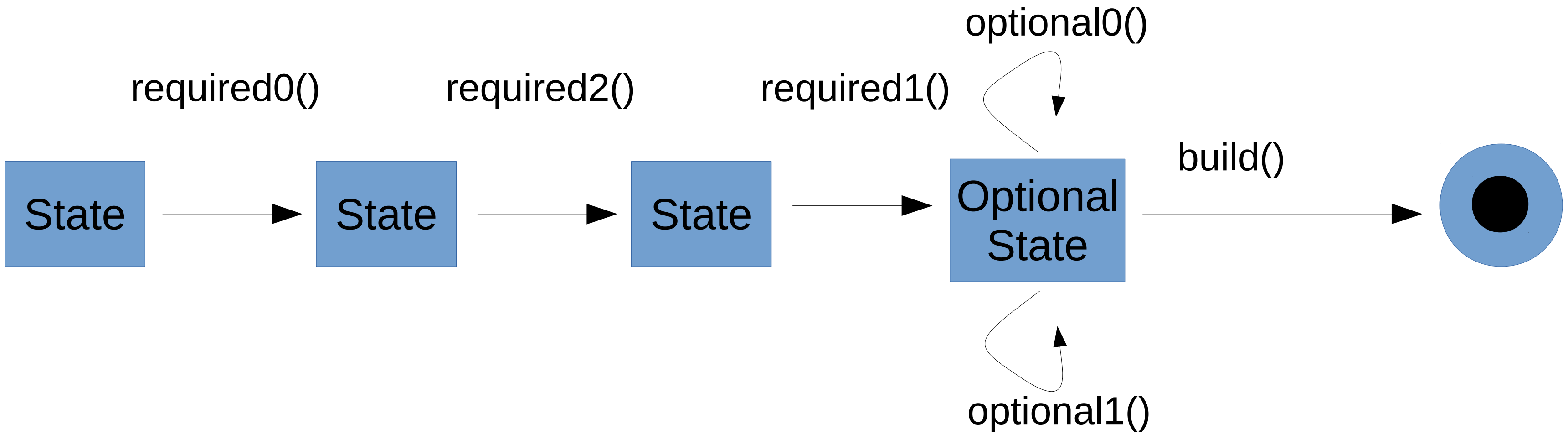


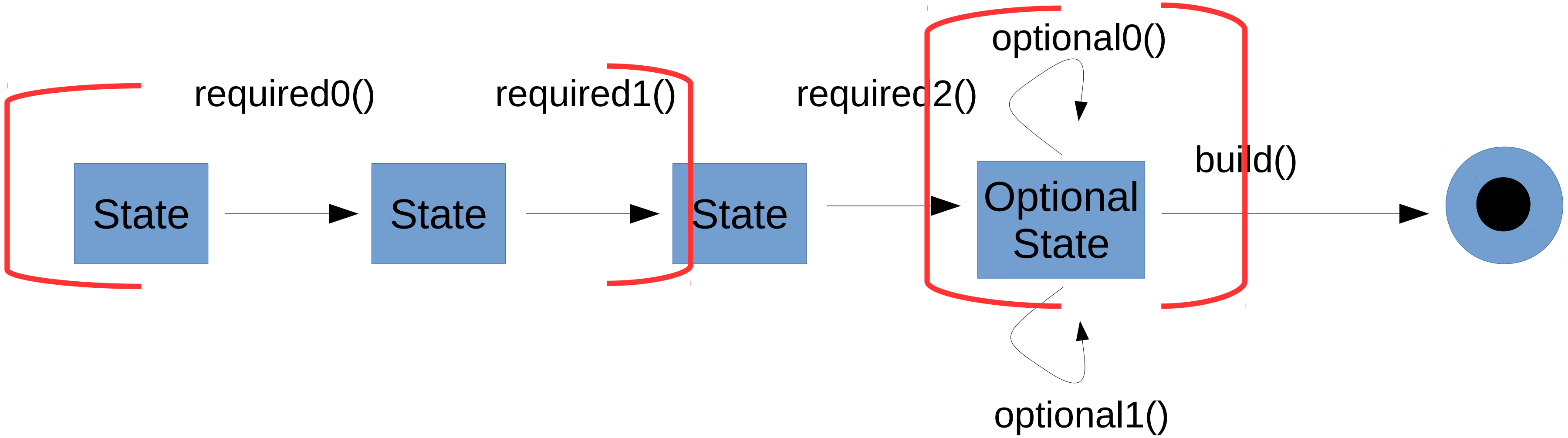
Intent Builder = Intent Builder Required Sequence + Intent Builder Optional State

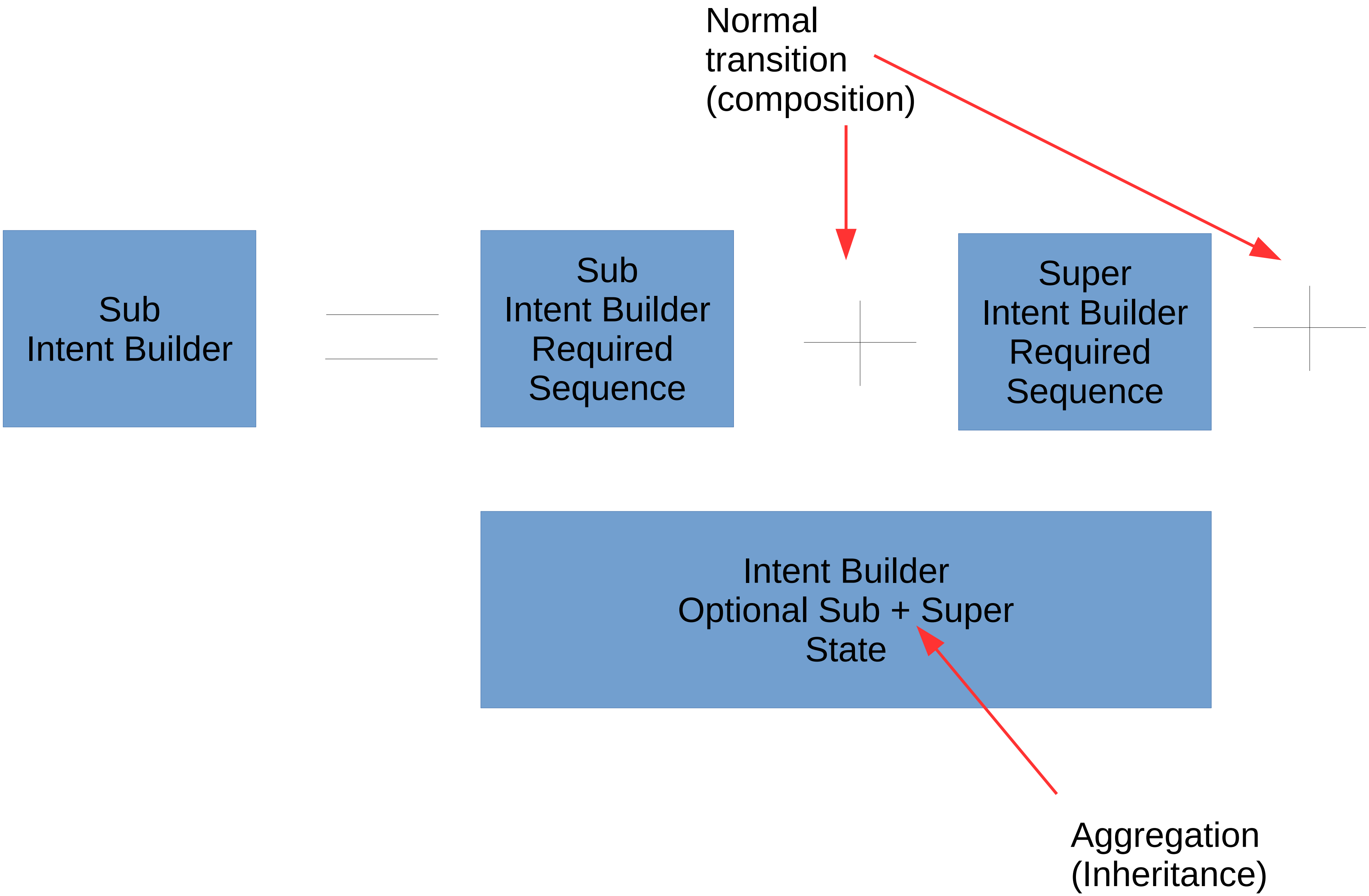
Intent Builder Required Sequence = State → State → State

Intent Builder Optional State = Optional State →


optional0()
required0()
required1()
required2()
build()
State
State
State
Optional State
optional1()


optional0()
required0()
required2()
required1()
build()
State
State
State
Optional State
optional1()


optional0()
required0()
required1()
required2()
build()
State
State
State
Optional State
optional1()



Normal
transition
(composition)

Sub
Intent Builder

=

Sub
Intent Builder
Required
Sequence

+

Super
Intent Builder
Required
Sequence

+

Intent Builder
Optional Sub + Super
State

Aggregation
(Inheritance)

Intent Builder
Optional Sub + Super
State

```
class SubIntentBuilderOptional
  extends SuperIntentBuilderOptional {

  //specific optional fields related methods
  public SubIntentBuilderOptional a(int a) {
    ...
  }
}
```

Optional states are combined by simple inheritance.

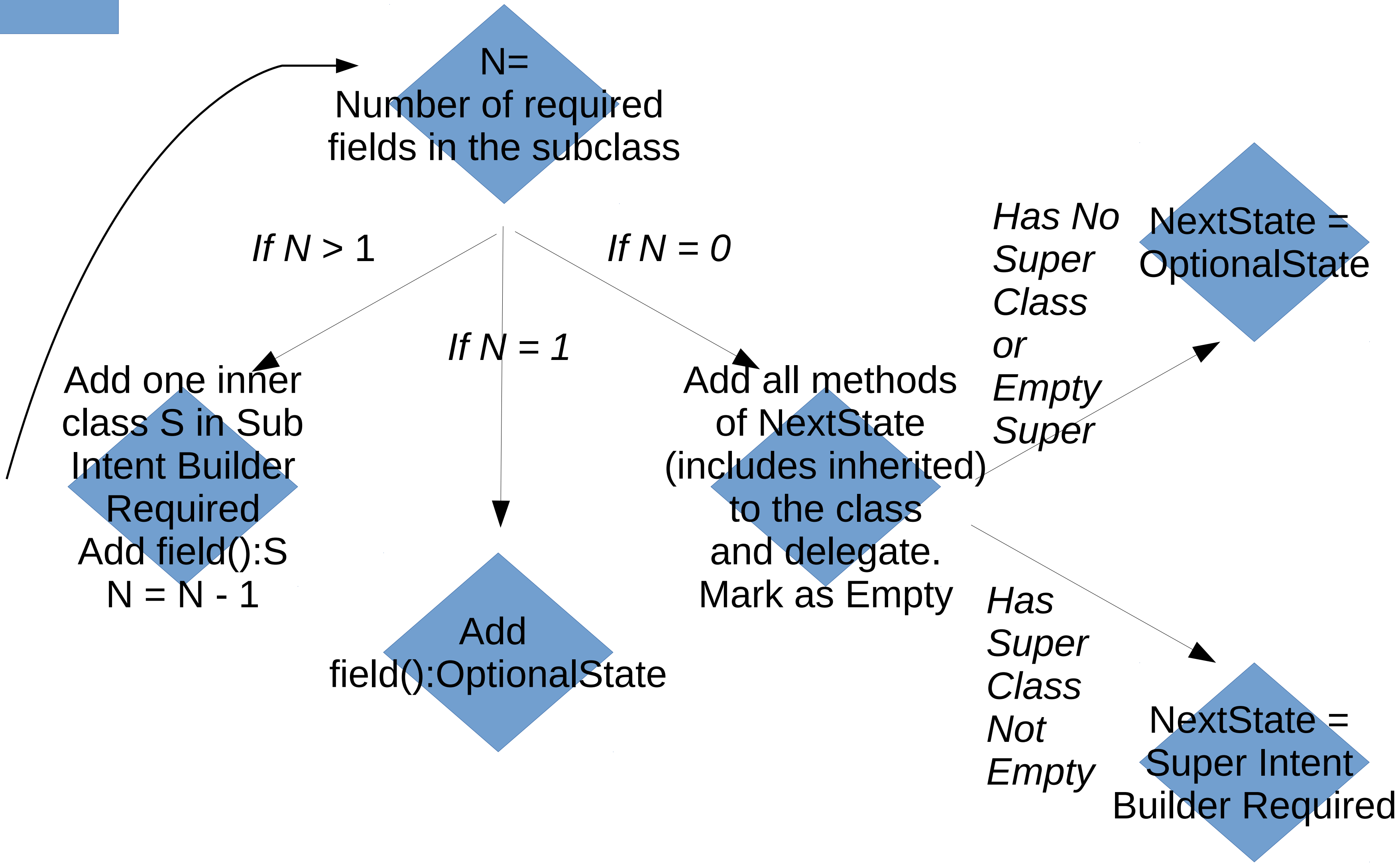
Intent Builder
Optional Sub + Super
State

```
class SubIntentBuilderOptional<T extends SubIntentBuilderOptional<T>>
  extends SuperIntentBuilderOptional<T> {

  //specific optional fields related methods
  public T a(int a) {
    ...
  }
}
```

This (known) trick allows to avoid co-variant overrides in future subclasses. All classes in the hierarchy will allows return of object of its own type (not it's super type).

Sub
Intent Builder
Required



N=
Number of required
fields in the subclass

If N > 1

If N = 1

If N = 0

Add one inner
class S in Sub
Intent Builder
Required
Add field():S
N = N - 1

Add
field():OptionalState

Add all methods
of NextState
(includes inherited)
to the class
and delegate.
Mark as Empty

*Has No
Super
Class
or
Empty
Super*

NextState =
OptionalState

*Has
Super
Class
Not
Empty*

NextState =
Super Intent
Builder Required

Tests & completeness:

both: both required parts and optional parts are present
req: only the required part is present, no optional fields
opt: only the optional part is present, no required field

super sub	both r.o	req r	opt o
both r1.o1	1	4	7
req r1	2	5	8
opt o1	3	6	9

1.r1.r.(o/o1)
2.r1.r.o1
3.r.(o/o1)
4.r1.r.o1
5.r1.r
6.r.o
7.r1.(o/o1)
8.r1.o
9.(o/o1)

Edge case: no required, no optional.

```
class IntentBuilder {
  IntentBuilderOptional ibo = new IntentBuilderOptional();
  IntentBuilderRequired req = new IntentBuilderRequired(ibo);
  public Intent build() {
   return req.build();
  }
}
class IntentBuilderRequired {
  IntentBuilderOptional ibo;
  public Intent build() {
    return ibo.build();
  }
}

class IntentBuilderOptional<T extends SubIntentBuilderOptional<T>> {
}
```

Edge case: required, no optional.

```
class IntentBuilder {
  IntentBuilderOptional ibo = new IntentBuilderOptional();
  IntentBuilderRequired req = new IntentBuilderRequired(ibo);
  public IntentBuilderOptional a(int a) {}
   return req.a(a);
  }

class IntentBuilderRequired {
  IntentBuilderOptional ibo;
  public IntentBuilderOptional a(int a) {
    return ibo;
  }
}

class IntentBuilderOptional<T extends SubIntentBuilderOptional<T>> {
}
```