CLASS ADAPTER PATTERN

THE TEXTBOOK EXAMPLE

WE'RE DONE!

Any Questions?

FINE, FINE, I'LL GIVE YOU A REAL-WORLD EXAMPLE

The Class Adapter is Baby-Boss-Friendly

HUH?

Have you Ever Dealt With A Boss Who Is:

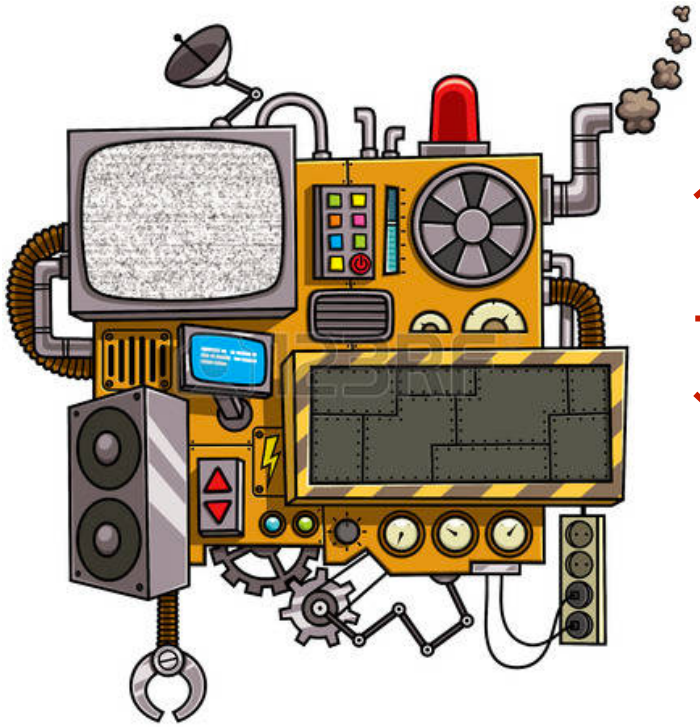
1) **Difficult**
2) **Stubborn**
3) **Fickle**
4) **Capricious**
5) **etc.** **(You get the idea)**

Then Listen Up. The Class Adapter May Save Your [redacted]*

* Profanity omitted for obvious reasons

CONSIDER THIS SCENARIO…

This is you.
You are a software engineer.



You maintain a complex piece of software built over the years.

Whose Interface Many Depend On.

You've been doing a spectacular job, and everything is running smoothly!

BUT THEN, ONE MORNING…

Baby boss shows up to work, half asleep…

“YOU’RE OKAY, BOSS?”

“Kiddo, I had a moment of genius last night [no surprise, really], and spent all night eating pizza and coding up new features for our system”

And, you’re like: “That’s great, boss!”

BUT THEN BABY BOSS GOES ON TO SAY…

“I NEED YOU TO INTEGRATE THEM”

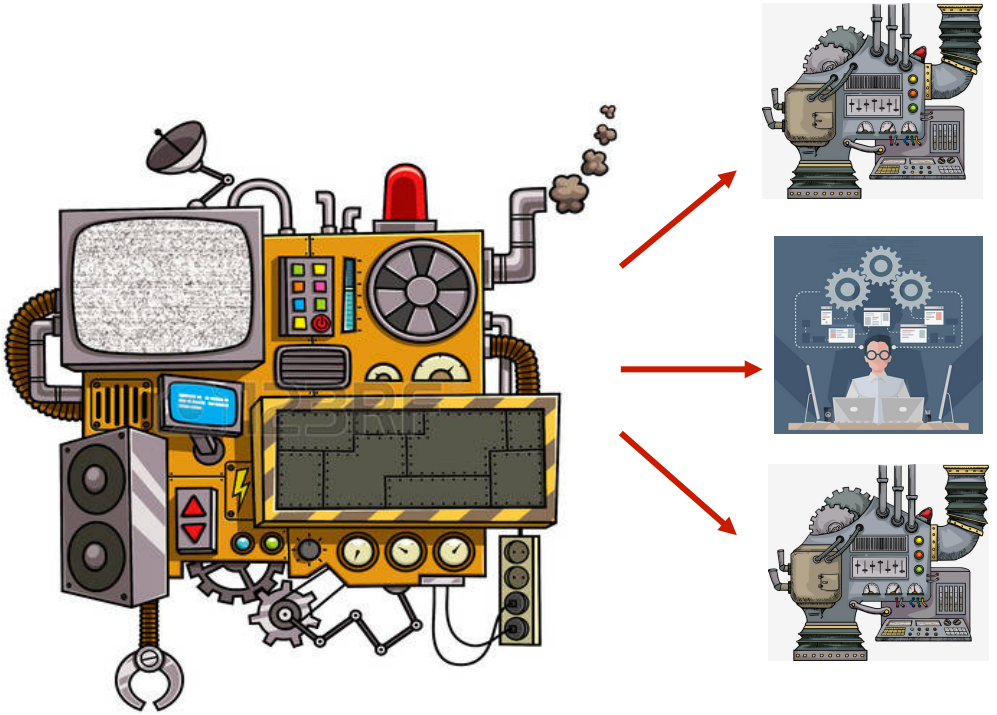
NO WORRIES, BOSS, I'VE GOT YOUR BACK

Until you realize that…

BABY BOSS DID NOT FOLLOW NAMING CONVENTIONS

The Interfaces Are Incompatible!

YOU TRY TO EXPLAIN



“Boss, so many external systems already call “command()”. We can’t just call “babyCommand()” without repercussions!

YOU EVEN TALK TO OTHER PEOPLE ON THE TEAM

But No One Seems To Care

ALL YOU'RE HEARING IS BABY BOSS SAYING…

1) "My methods are brilliantly named. Don't touch them!"

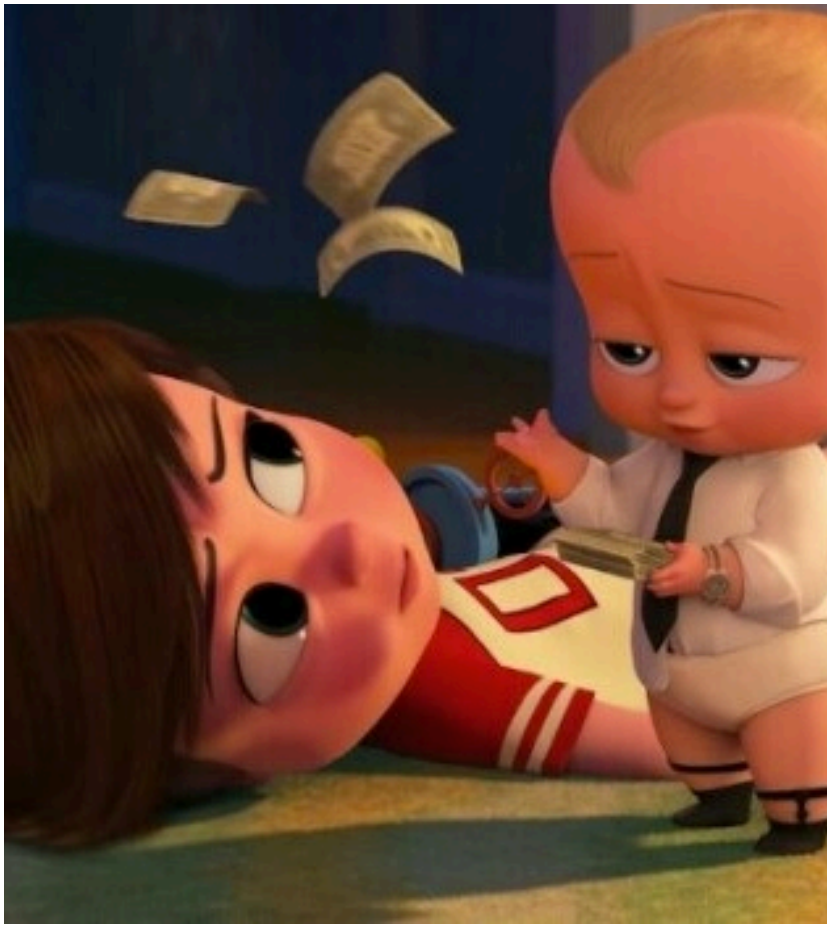
2) "Don't be such a baby"

3) "Just get it done"

4) "I don't care what it costs"

(but really, he does)

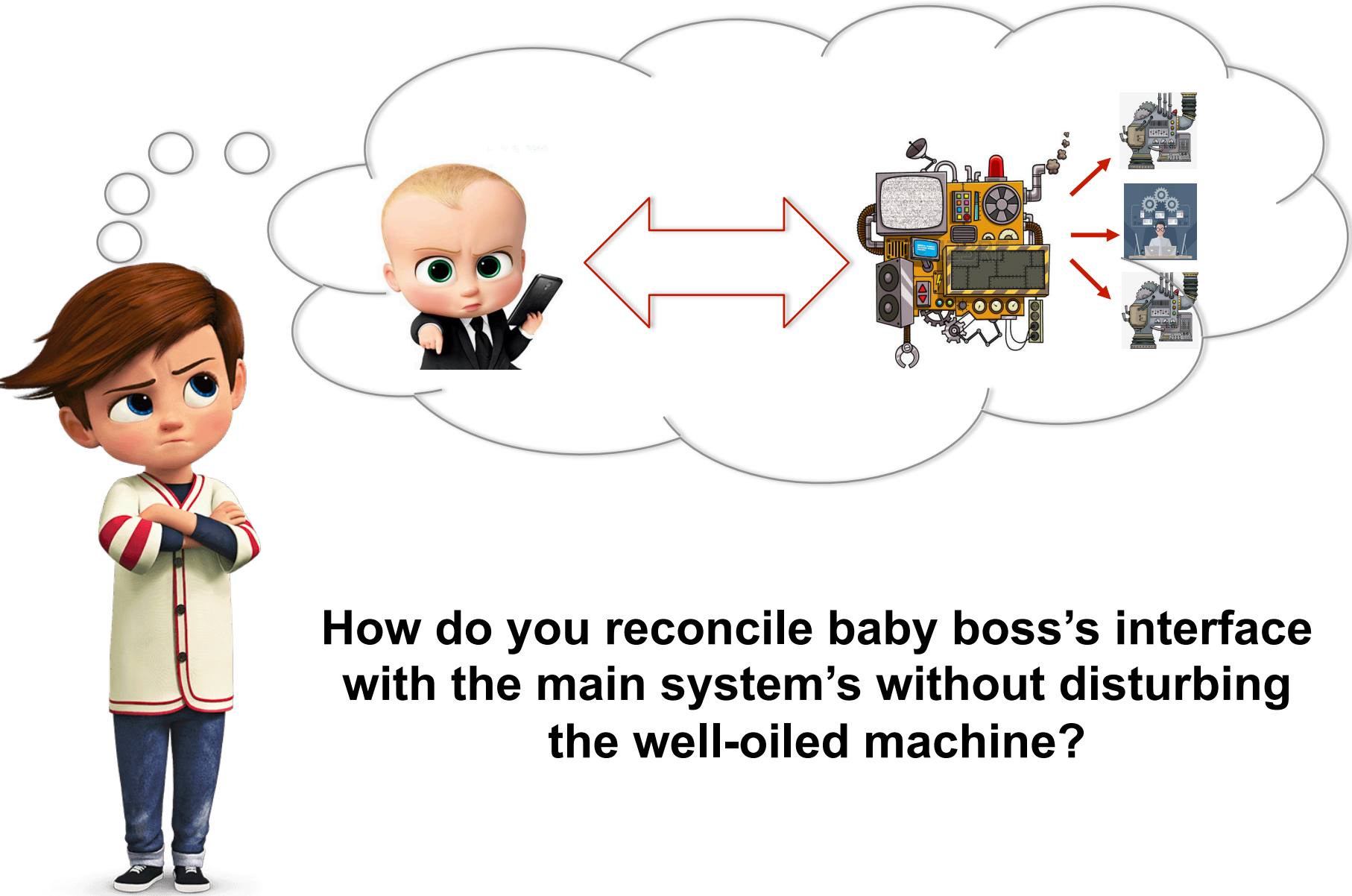
5) "I'll be sleeping, but have it on my desk at 4:34 am tomorrow morning"



If you think this is fictional, think again.

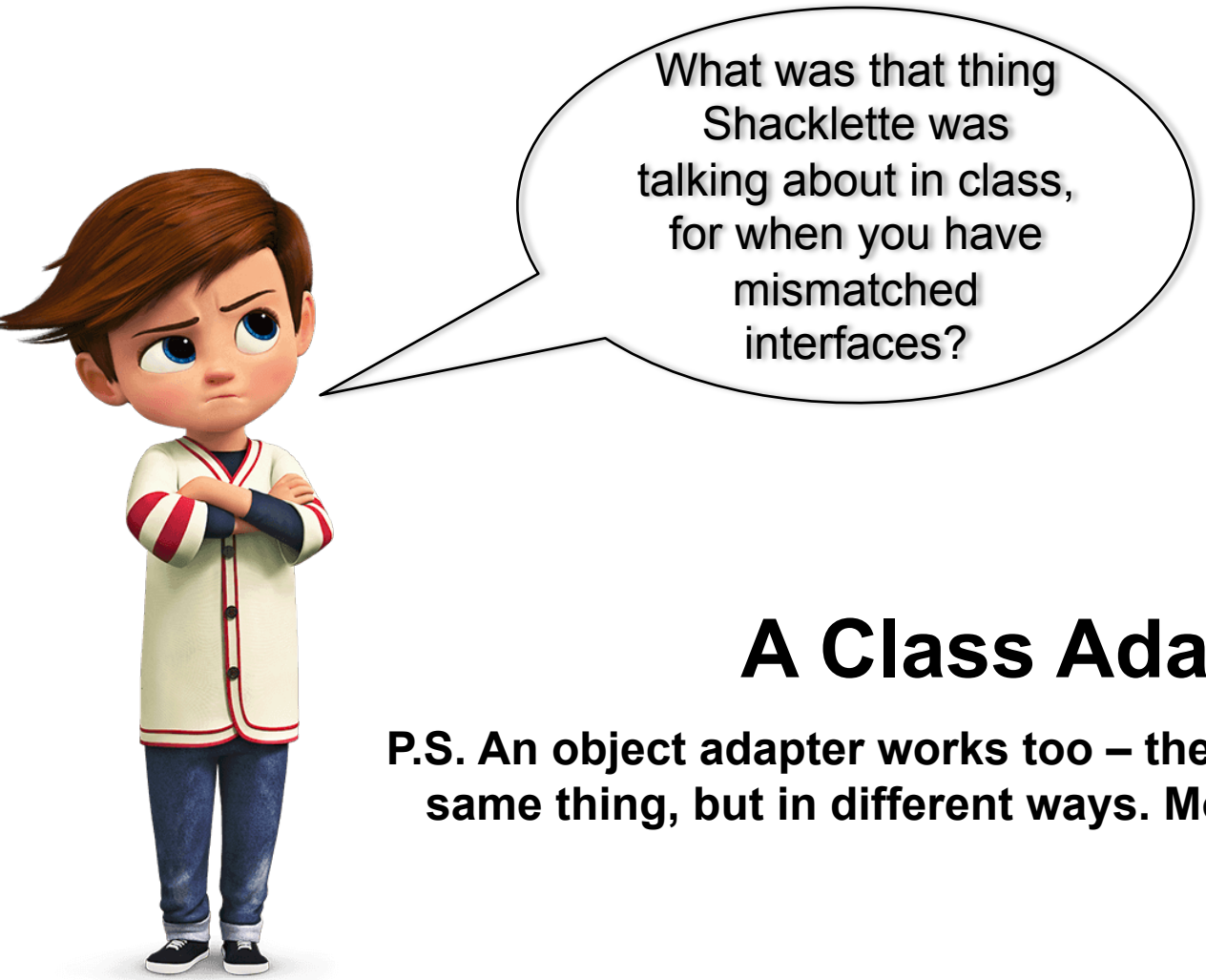
Sadly, it's all too so very real!

YOU HAVE A CONUNDRUM



How do you reconcile baby boss's interface with the main system's without disturbing the well-oiled machine?

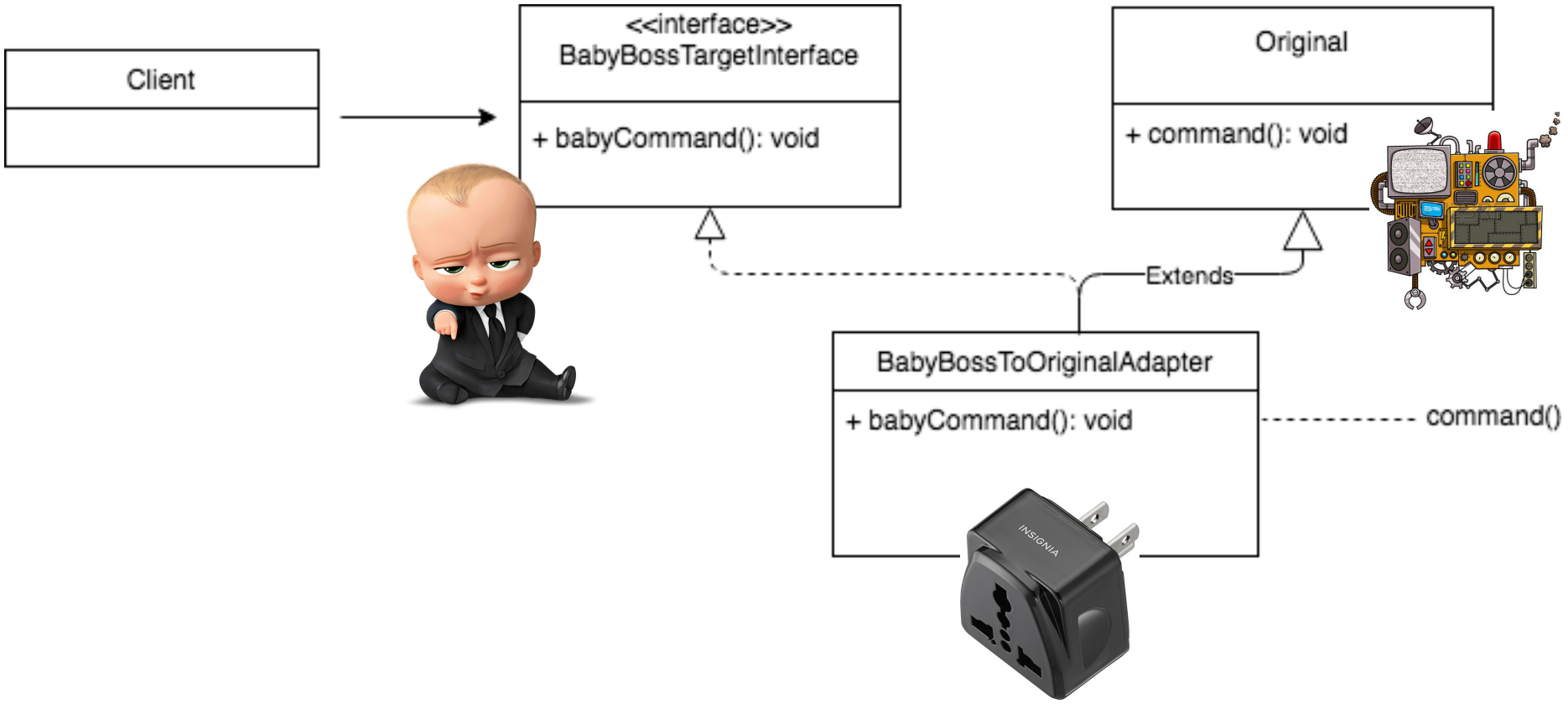
AND THEN IT DAWNS UPON YOU



A Class Adapter!

P.S. An object adapter works too – they both really achieve the same thing, but in different ways. More on that in just a bit.

ARCHITECTURE



NOW SOME CODE…

In C++

(…gasp…)

WAIT

Before you start booing me for picking the most esoteric language ever created (Sorry, Paul)

LET ME EXPLAIN...

Multiple inheritance is a key feature of the class adapter pattern

We need C++ for this one.

Recall that C++ (and python) supports multiple inheritance.

Java / Ruby do not.

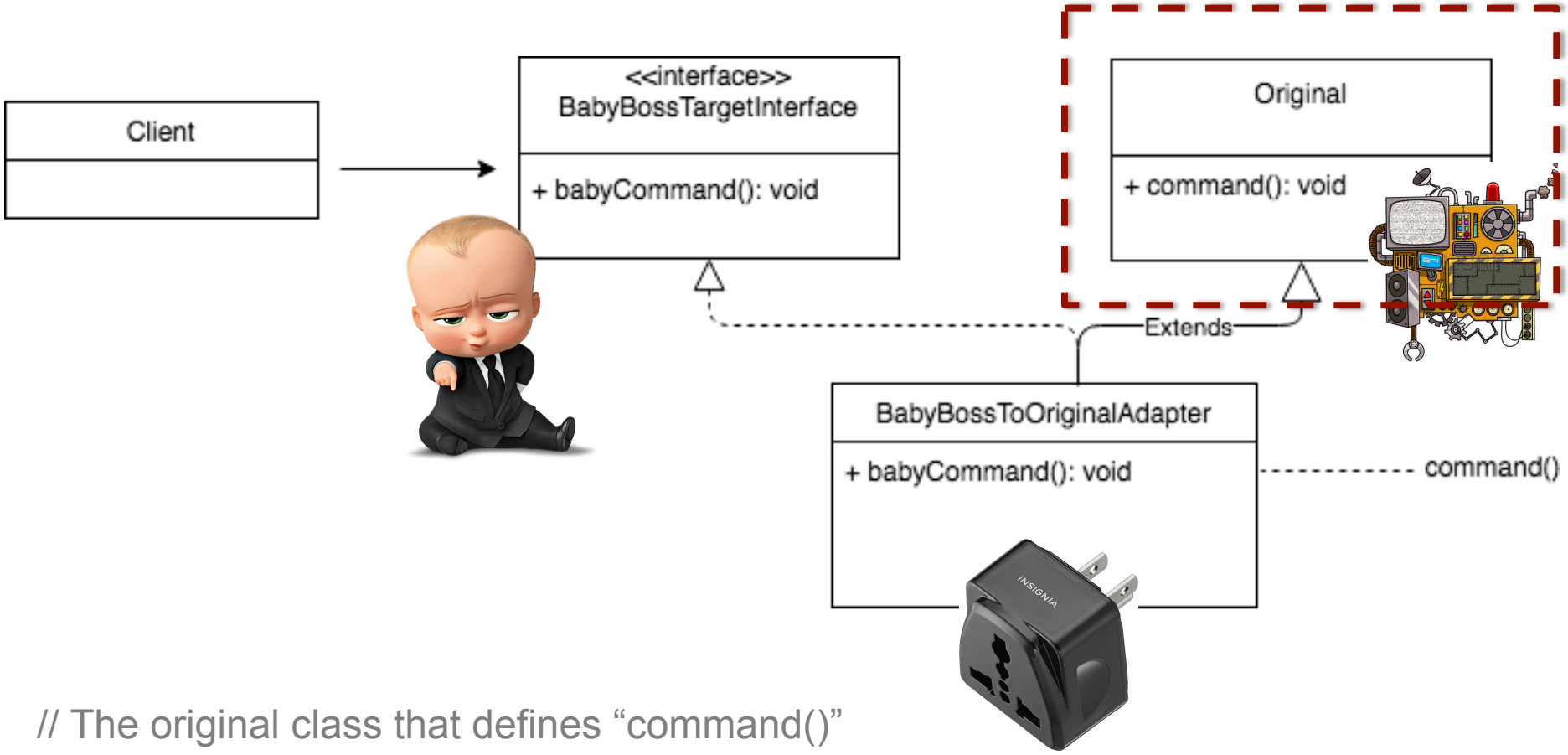
HOW AN ADAPTER DOES ITS MAGIC

The class adapter "adapts" interfaces via multiple inheritance

The object adapter "adapts" interfaces via composition

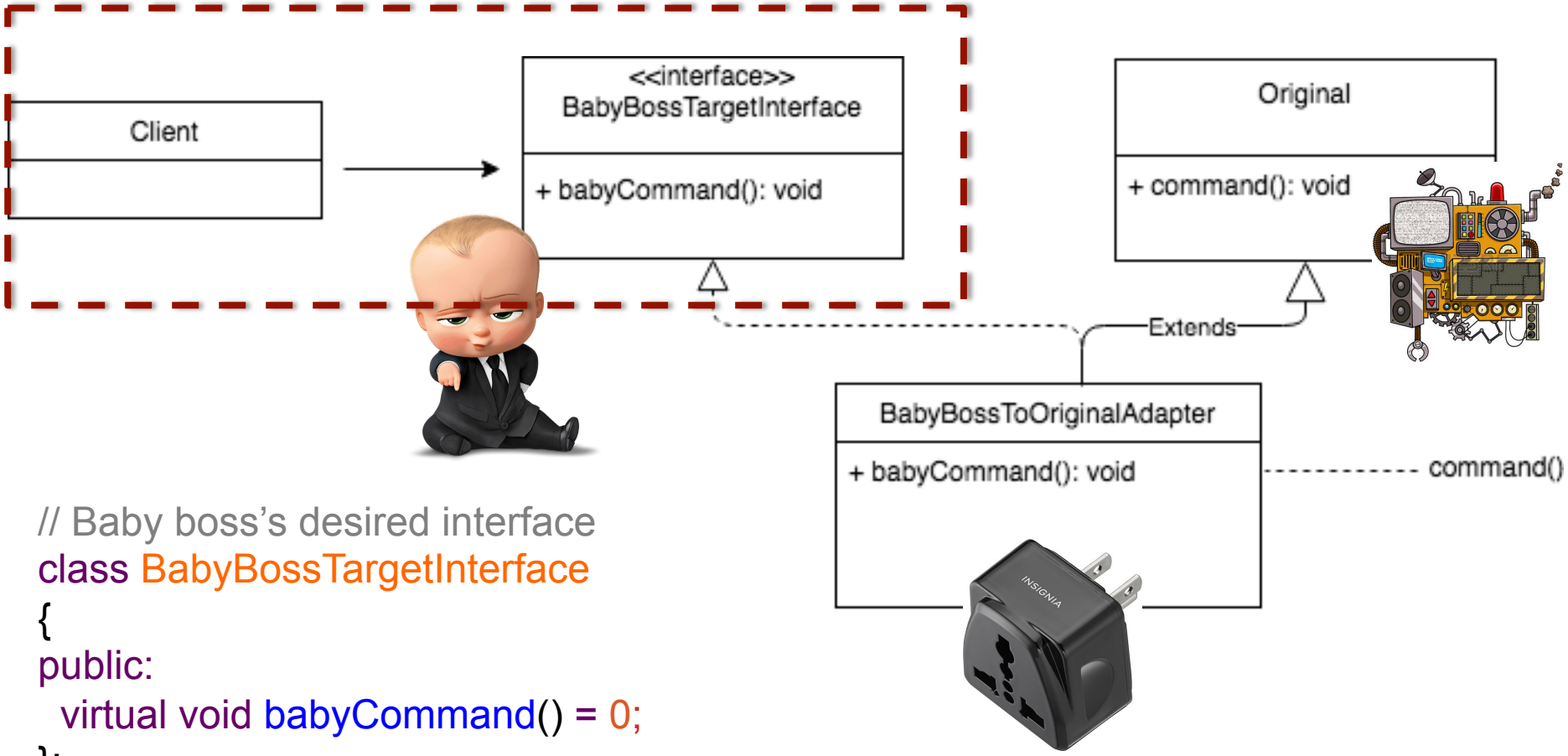
But, I won't steal my colleague's thunder.

Stay tuned for the next presentation.



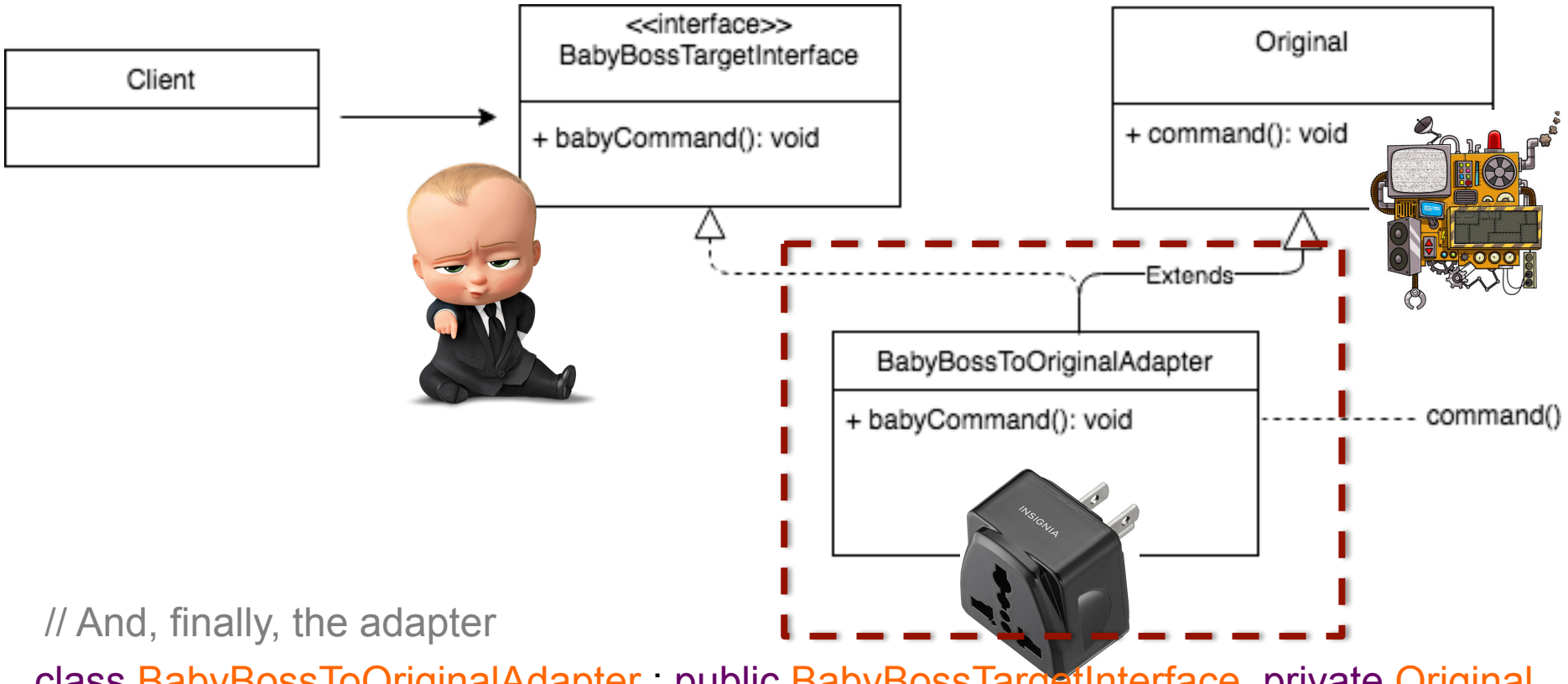

```
// The original class that defines “command()”
class Original
{
public:
  Original(){}
  void command()
  {
    std::cout << "Calling the original command" << std::endl;
  }
};
```

Recall that many external systems depend on “command()”, so changing the interface in the Original class is risky business




```
// Baby boss's desired interface
class BabyBossTargetInterface
{
public:
  virtual void babyCommand() = 0;
};

// Client knows nothing about the original class and "command()"
// yet will be using "command()" via the interface baby boss mandated
int main()
{
  BabyBossTargetInterface * babyBoss = new BabyBossToOriginalAdapter();
  babyBoss -> babyCommand();
}
```




```
// And, finally, the adapter
class BabyBossToOriginalAdapter : public BabyBossTargetInterface, private Original
{
public:
  BabyBossToOriginalAdapter() {}
  virtual void babyCommand()
 {
    std::cout << “From babyCommand()” << std::endl;
    command();
 }
};
```

Notice the target interface is publically inherited, whereas Original’s methods are only visible to the adapter client

The client can call “babyCommand()”, and not “command()”


NOW, FOR REAL, ANY QUESTIONS?