Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

Programmazione Concorrente e Distribuita
Sincronizzazione di thread

Luigi Lavazza
Dipartimento di Scienze Teoriche e Applicate
luigi.lavazza@uninsubria.it

Esercizio 1 – Irish pub

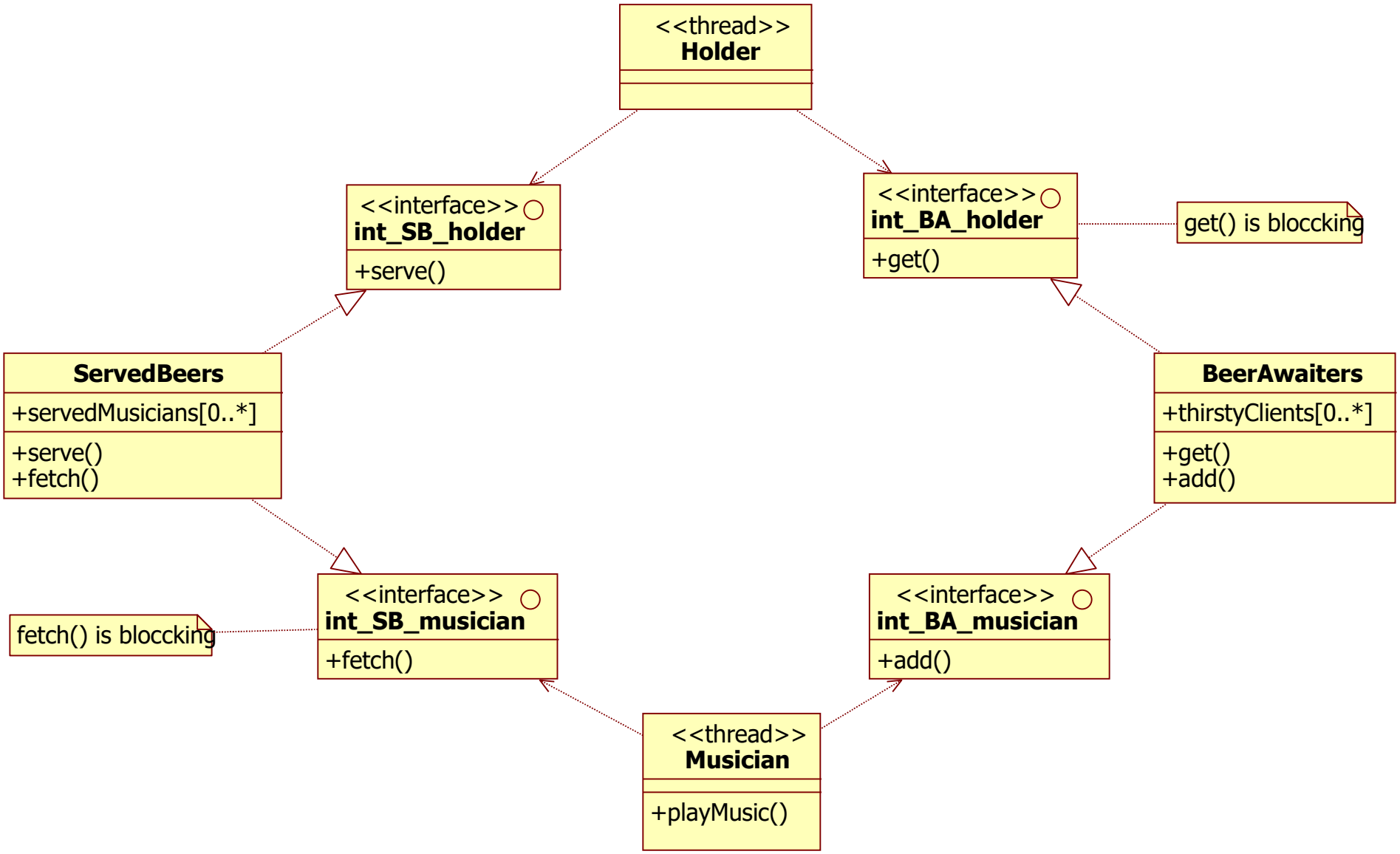
- In un Irish pub i clienti che portano uno strumento musicale e suonano ricevono delle birre in omaggio dal proprietario ogni volta che fanno una pausa (smettendo di suonare).
- Il proprietario offre M birre e non di più.
- Ad ogni pausa, il cliente musicista aspetta di vedere se il proprietario gli porta la birra.
  - Se sì, se la beve e poi ricomincia a suonare
  - Se no, dopo un certo tempo ricomincia comunque a suonare
- Il cliente musicista suona un certo numero di volte (diverso per ogni musicista) e poi va a casa
- Il proprietario è sempre in attesa di vedere se qualche musicista ha sete (finché non finisce le birre gratis).

- Il problema consiste nel sincronizzare il thread Holder e i thread Musician in base all'evento consistente nel fatto che un cliente musicista ha smesso di suonare e ha voglia di birra

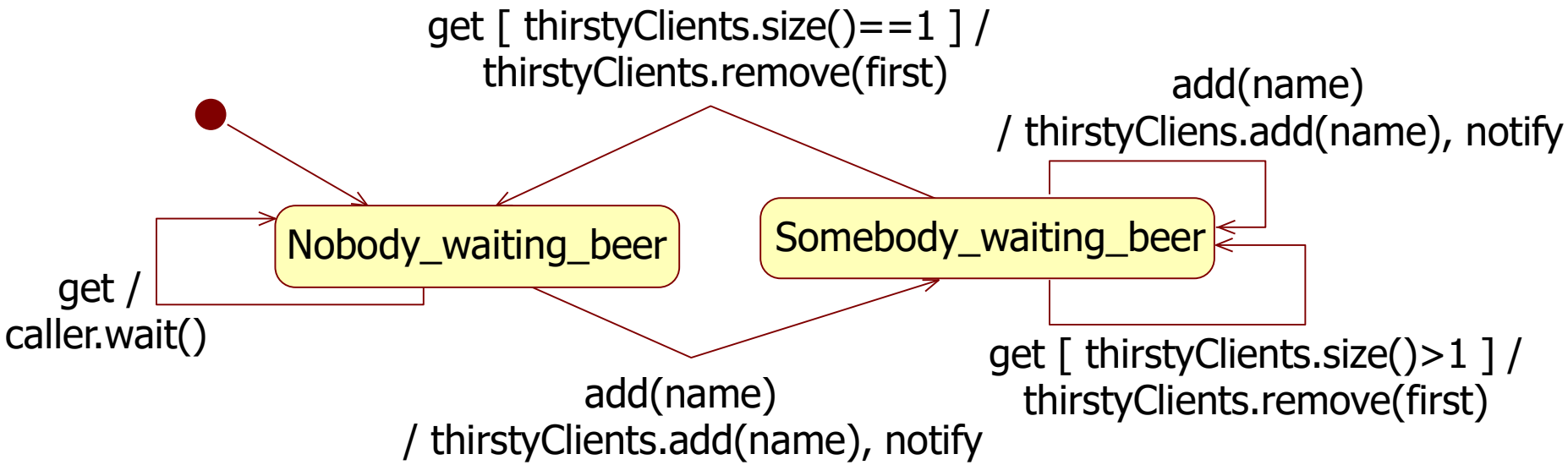
Caratteristica del problema

- Il proprietario aspetta un evento (pausa) causato dai suonatori
- I suonatori aspettano un evento (birra servita) causato dal proprietario
  - Questo evento puo` anche non verificarsi (quando le birre gratis sono finite)

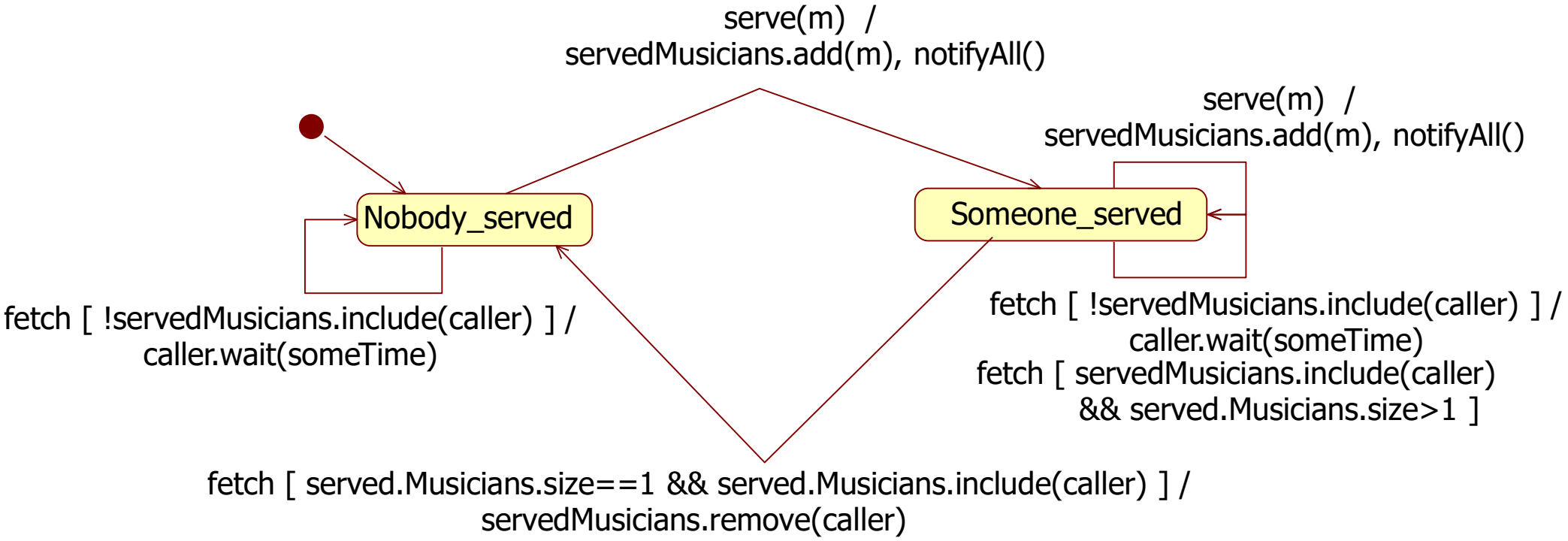
Irish Pub


<<thread>>
Holder
<<interface>>
int_SB_holder
+serve()
<<interface>>
int_BA_holder
+get()
get() is bloccking
ServedBeers
+servedMusicians[0..*]
+serve()
+fetch()
BeerAwaiters
+thirstyClients[0..*]
+get()
+add()
<<interface>>
int_SB_musician
+fetch()
fetch() is bloccking
<<interface>>
int_BA_musician
+add()
<<thread>>
Musician
+playMusic()

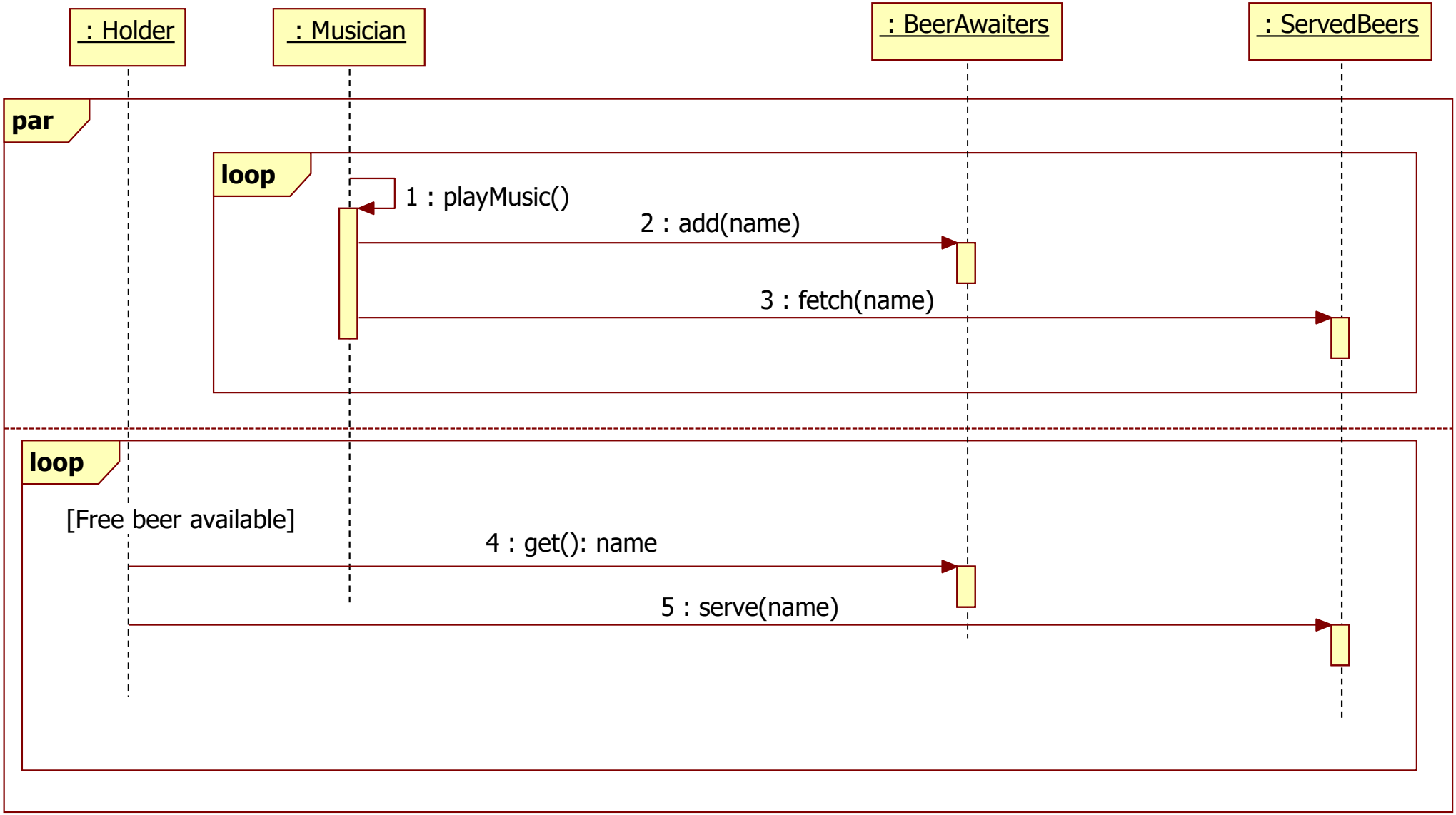
BeerAwaiters – state diagram



BeerServed – state diagram



Irish Pub

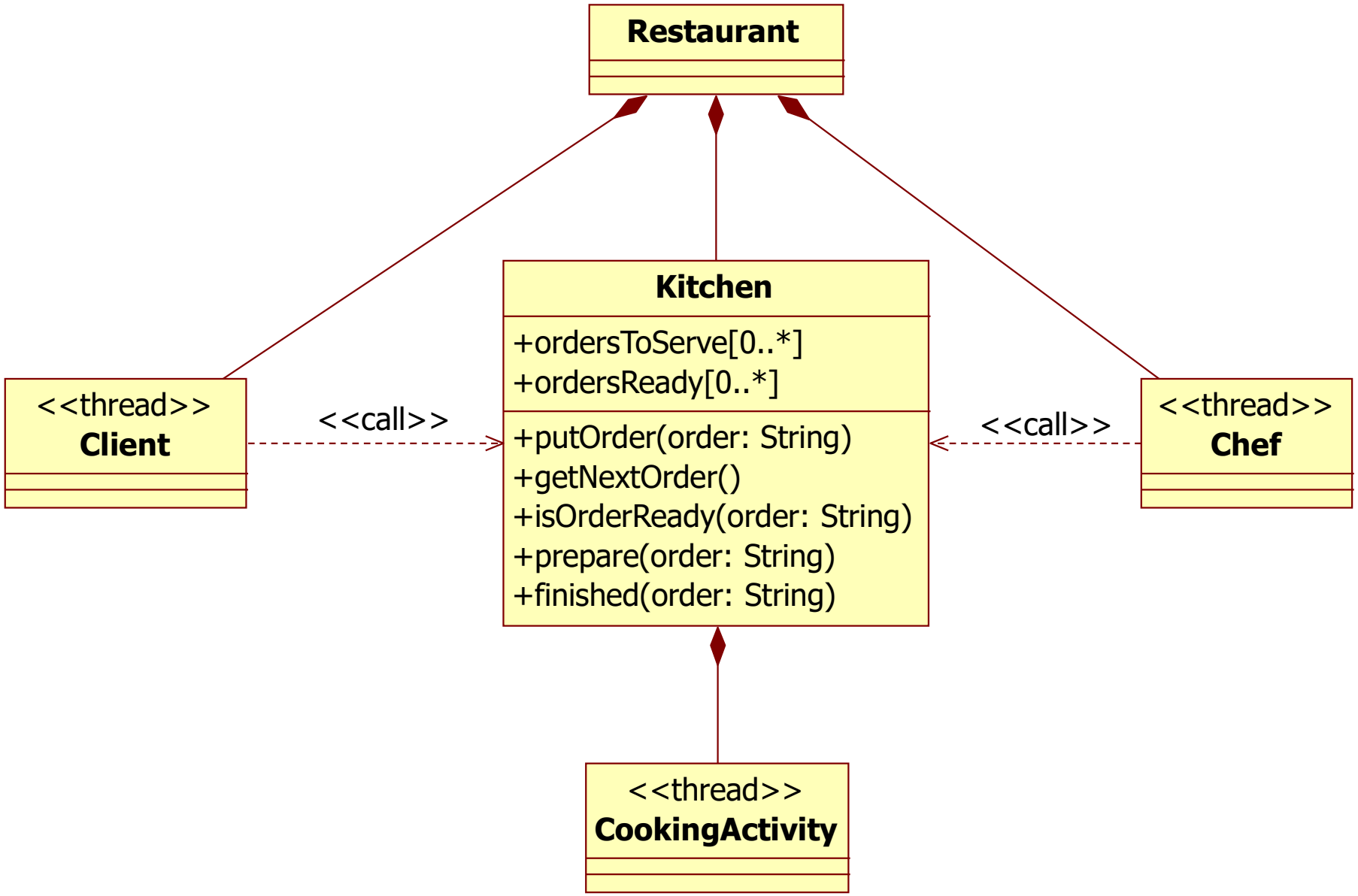

: Holder
: Musician
: BeerAwaiters
: ServedBeers
par
loop
1 : playMusic()
2 : add(name)
3 : fetch(name)
loop
[Free beer available]
4 : get(): name
5 : serve(name)

Esercizio 2 – Ristorante

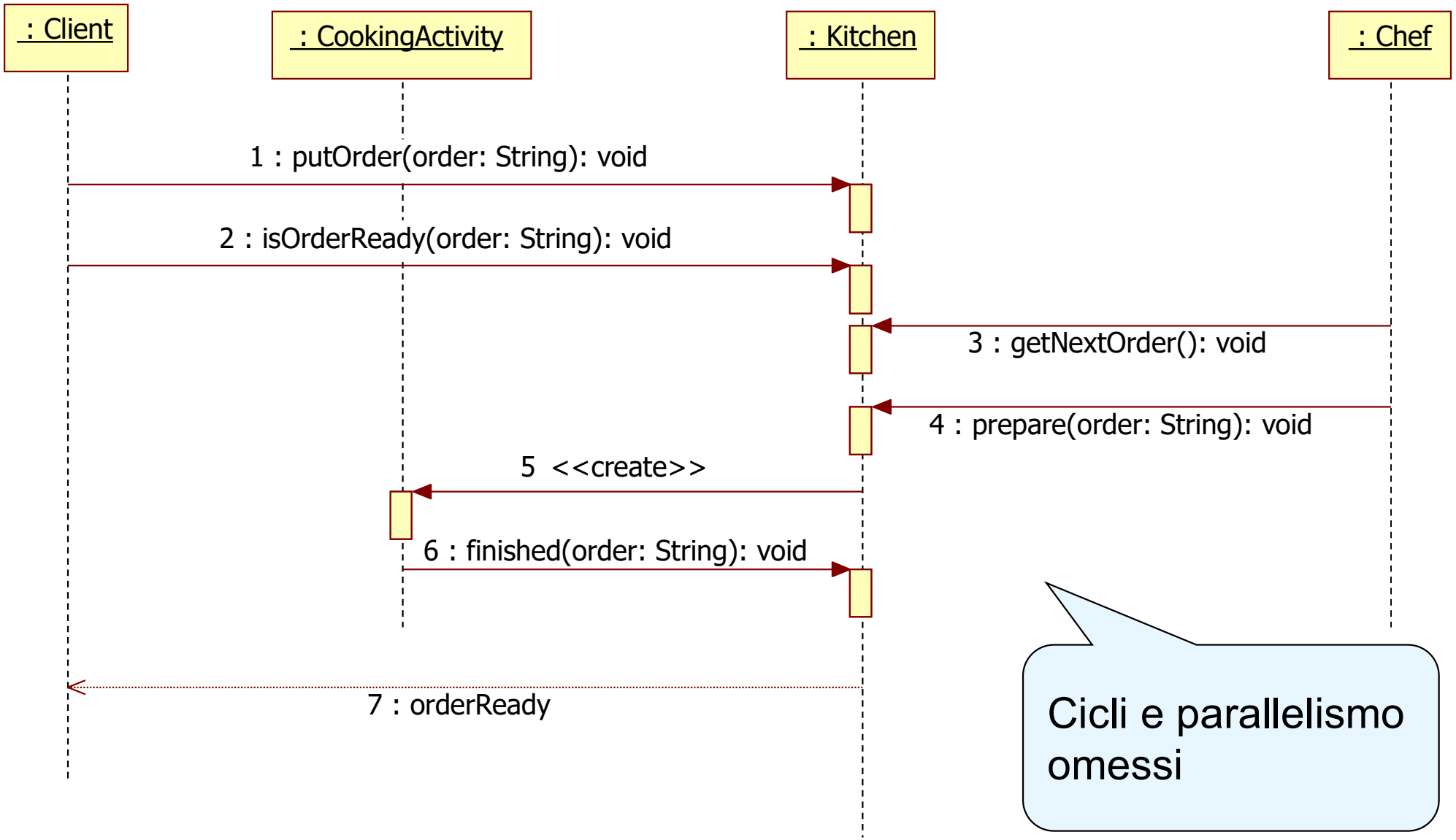
- In un ristorante entrano diversi clienti.
- Ciascun cliente fa un'ordinazione
- Le ordinazioni arrivano allo chef, che può gestire al massimo M ordinazioni contemporaneamente. Quelle in eccesso sono accodate.
- Lo chef non fa altro che servire le ordinazioni presenti. Quando non ce ne sono, aspetta che ne arrivino.
- Il cliente ordina, aspetta di essere servito, mangia e se ne va.

- Si programmi un sistema che si comporta come descritto.

Ristorante


Restaurant
Kitchen
+ordersToServe[0..*]
+ordersReady[0..*]
+putOrder(order: String)
+getNextOrder()
+isOrderReady(order: String)
+prepare(order: String)
+finished(order: String)
<<thread>>
Client
<<call>>
<<call>>
<<thread>>
Chef
<<thread>>
CookingActivity

Ristorante


: Client
: CookingActivity
: Kitchen
: Chef
1 : putOrder(order: String): void
2 : isOrderReady(order: String): void
3 : getNextOrder(): void
4 : prepare(order: String): void
5 <<create>>
6 : finished(order: String): void
7 : orderReady
Cicli e parallelismo omessi

Estensione – Ristorante con capienza massima

- Rivedere il programma precedente per fare in modo che nel ristorante non possano mai esserci più di N client.

Esercizio 3 – Il barbiere addormentato

Il barbiere addormentato

- Un negozio di barbiere con
  - 1 barbiere
  - 1 poltrona per il taglio
  - N sedie per clienti in attesa
- Se non vi sono clienti nel negozio il barbiere dorme sulla poltrona
- Un cliente che entra nel negozio in cui c'è solo il barbiere che dorme lo sveglia; il barbiere fa accomodare il cliente sulla poltrona e lo serve.
- I clienti che entrano trovando la poltrona occupata si mettono in attesa su una sedia libera; se non ci sono sedie libere se ne vanno.
- Quando il barbiere finisce di servire un cliente serve il primo cliente in attesa; se non ce ne sono, torna a dormire.

Esercizio 4 – Il problema dei lettori e degli scrittori

- Diversi lettori e almeno uno scrittore accedono a un insieme di dati condiviso.
  - I lettori si limitano a leggere i dati.
  - Gli scrittori modificano i dati.
- Si vuole che in ogni momento una sola di queste situazioni sia verificata:
  - Uno scrittore sta accedendo ai dati.
    - Un solo scrittore, nessun lettore
  - Un insieme di lettori sta accedendo ai dati
    - Nessuno scrittore, un numero qualunque di lettori
  - Nessun thread sta accedendo ai dati
    - Nessun lettore e nessuno scrittore

Cosa bisogna assicurare

- Mutua esclusione con le regole date
- Evitare starvation per scrittori
  - I lettori potrebbero avvicendarsi alla lettura impedendo indefinitamente allo scrittore di accedere
  - In altre parole, c'è sempre qualcuno che legge e nessuno può mai scrivere.