Tampereen yliopisto

# C++:n *Standard Template Library* (STL)

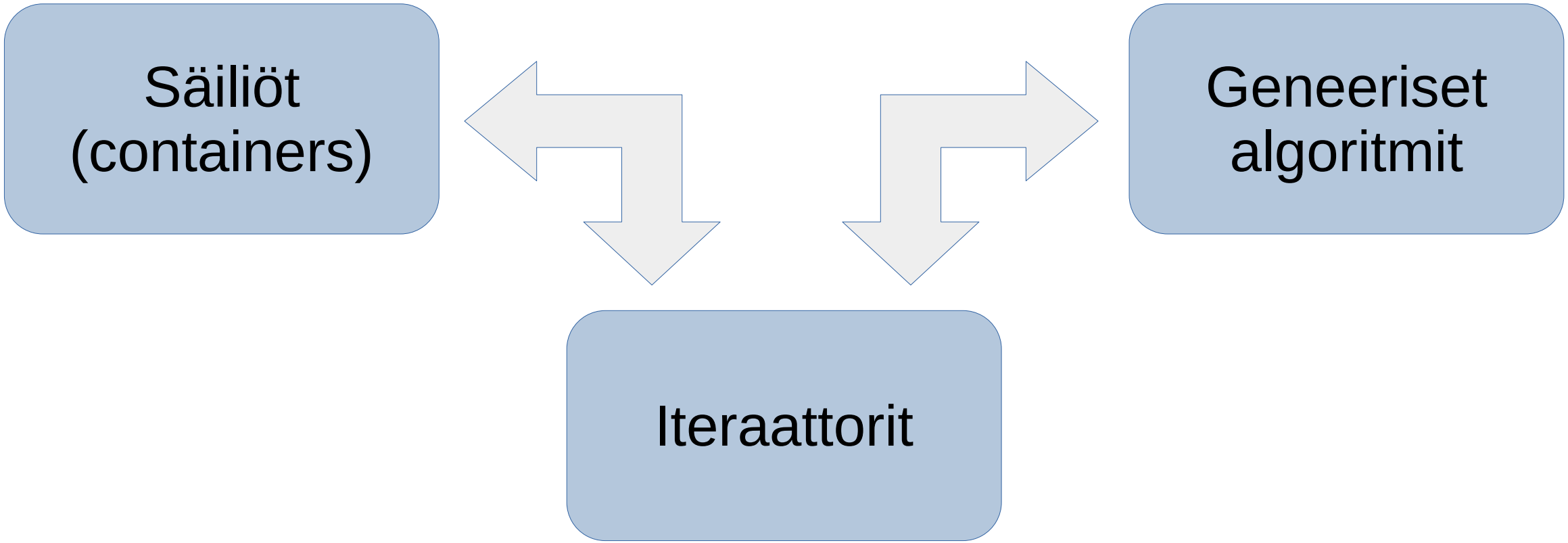
*COMP.CS.300 Tietorakenteet ja algoritmit 1*

*Matti Rintala (matti.rintala@tuni.fi)*

# Valmis kirjasto vs oma toteutus

- Itse toteutettu tietorakenne/algoritmi:
  - Miten operaatiot *toteutetaan?*
- Valmis tietorakenne/algoritmikirjasto (STL):
  - Toteutus piilotettu
  - Miten operaatioita *käytetään?* (Rajapinnat)
  - Miten valita *sopiva* tietorakenne/algoritmi?
  - Miten valita *tehokas* tietorakenne/algoritmi?
  - Miten *yhdistellä* valmiita tietorakenteita/algoritmeja?
  - Miten *säätää/mukauttaa* valmiiden operaatioiden toimintaa?

# STL:n (tietorakenne)osat



Tampereen yliopisto

# STL:n (tietorakenne)osat

Perusoperaatiot:

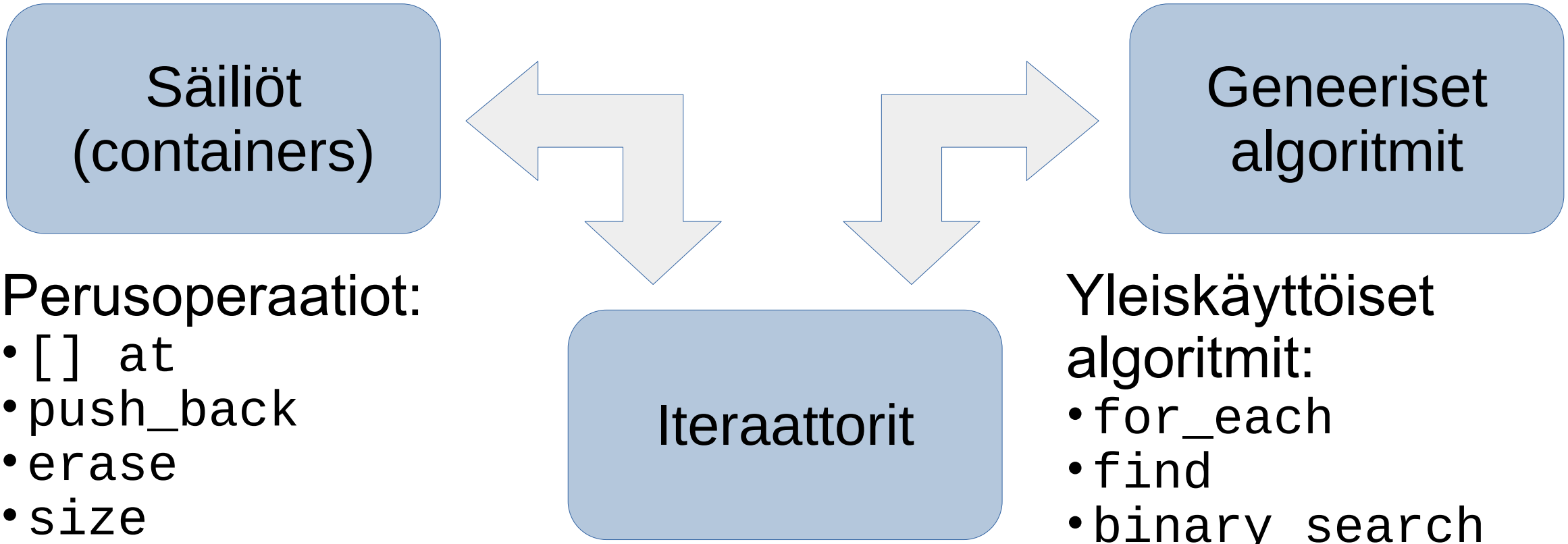
- `[] at`
- `push_back`
- `erase`
- `size`
- `clear`
- `...`

# STL:n (tietorakenne)osat

Yleiskäyttöiset algoritmit:

- `for_each`
- `find`
- `binary_search`
- `set_symmetric_difference`
- `transform_reduce`
- `...`

# STL:n (tietorakenne)osat




Perusoperaatiot:

- `[] at`
- `push_back`
- `erase`
- `size`
- `clear`
- `...`

Yleiskäyttöiset algoritmit:

- `for_each`
- `find`
- `binary_search`
- `set_symmetric_difference`
- `transform_reduce`
- `...`

# STL ja asymptoottinen tehokkuus

- STL:n operaatioiden tehokkuus luvattu asymptoottisesti
  - Usein lupaus O-notaatio ("ei hitaampi")
  - Joskus myös keskimääräinen tehokkuus tai Θ-notaatio

# STL ja asymptoottinen tehokkuus

- STL:n operaatioiden tehokkuus luvattu asymptoottisesti
  - Usein lupaus O-notaatio ("ei hitaampi")
  - Joskus myös keskimääräinen tehokkuus tai Θ-notaatio
- Mutta mikä on *"n"?* Riippuu tilanteesta:
  - Alkioiden lukumäärä
  - Tietyn alkio-osajoukon lukumäärä
  - Joskus useita muuttujia (O(*m*n*))

Tampereen yliopisto

# STL:n säiliöt (containers)

*COMP.CS.300 Tietorakenteet ja algoritmit 1*

*Matti Rintala (matti.rintala@tuni.fi)*

# STL:n säiliöt (containers)

- Sarjat (sequences)
  - Alkioiden järjestys *itse valittavissa*
  - Alkiot löytyvät järjestyksen perusteella
- Assosiatiiviset säiliöt
  - *Säiliö päättää* alkioiden järjestyksen (voi olla myös määrittelemätön)
  - Alkiot löytyvät *hakuavaimen* perusteella
- (Säiliösovittimet (adaptors) `stack`, `queue`, `priority_queue`)

# STL:n sarjat (sequences)

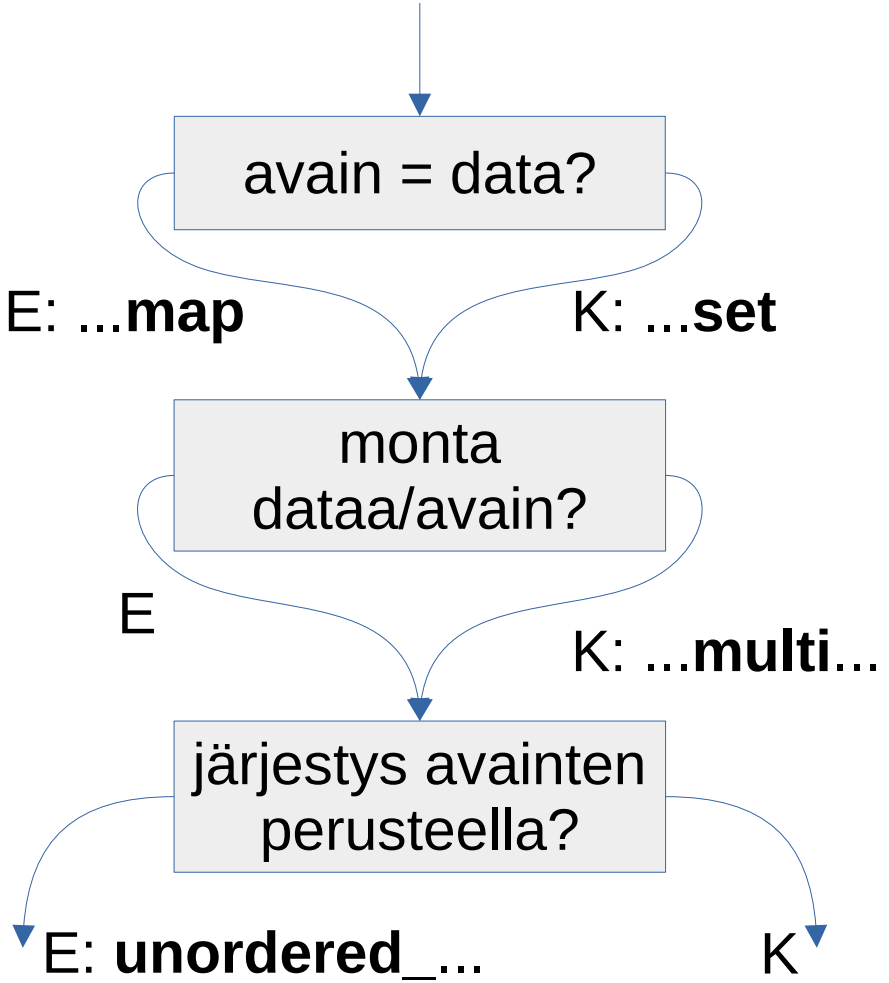
- `vector`, `deque`, `list`
- `(array, forward_list)`
- Alkioiden järjestys itse valittavissa
- Haku *indeksoimalla* (järjestysnumero) tai *iteroimalla* (läpikäynti järjestyksessä)
- Lisäys/poisto annetusta *kohdasta* (iteraattori)

# STL:n assosiatiiviset säiliöt

- Järjestetyt `map`, `set`
  - Järjestys hakuavaimen mukainen
- `unordered_map/_set`
  - Järjestys *määrittelemätön,* voi vaihtua koska vain!
- Monta alkiota per hakuavain: (unordered_)**multi**map/set
- Poisto annetusta *kohdasta* (iteraattori) (kohta löytyy haulla)

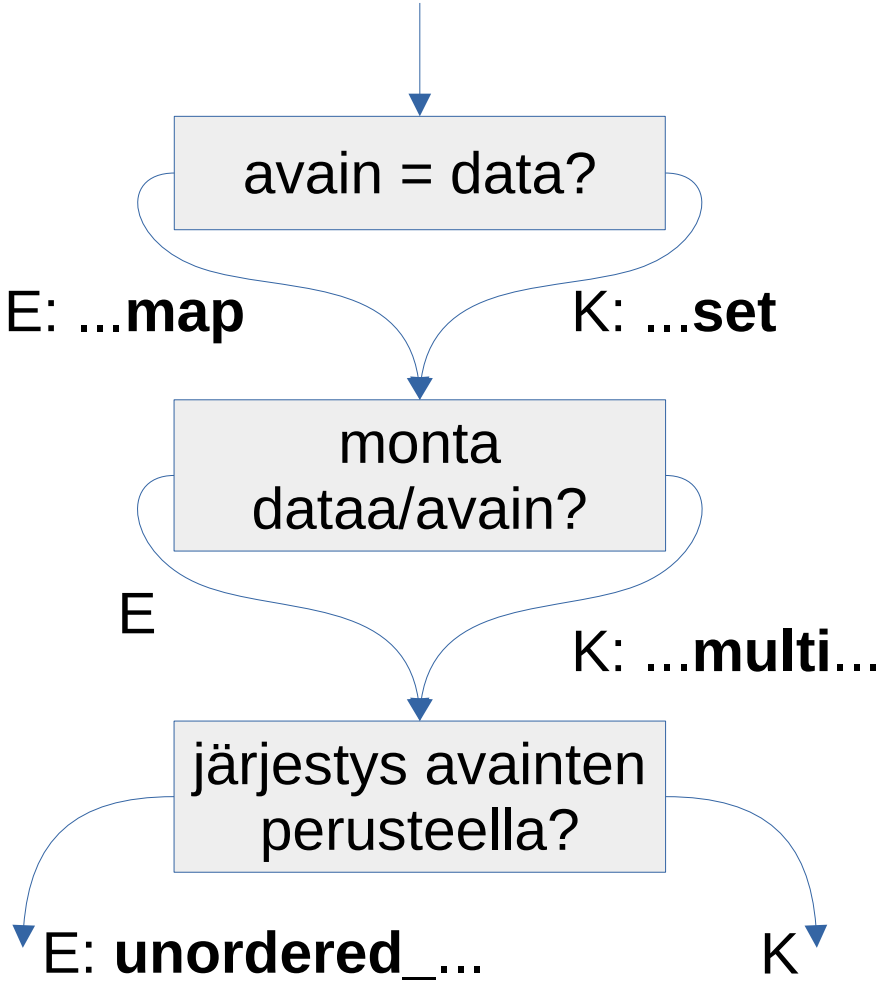
Tampereen yliopisto

# STL:n assosiatiiviset säiliöt



# STL:n assosiatiiviset säiliöt

- Järjestetyt `map`, `set`
  - Järjestys hakuavaimen mukainen
- `unordered_map/_set`
  - Järjestys *määrittelemätön,* voi vaihtua koska vain!
- Monta alkiota per hakuavain: (unordered_)**multi**map/set
- Poisto annetusta *kohdasta* (iteraattori) (kohta löytyy haulla)



Tampereen yliopisto

# STL:n säiliöiden tehokkuus

*COMP.CS.300 Tietorakenteet ja algoritmit 1*

*Matti Rintala (matti.rintala@tuni.fi)*

Tampereen yliopisto

# Säiliöiden (asymptoottinen) tehokkuus

- Asymptoottinen tehokkuus määrätty (yleensä yläraja O)
- Rajapinnat keskenään samantapaiset, *erot tehokkuudessa*
- Jos operaatio "tehoton", se voi puuttua säiliöstä
- Säiliön valinta:
  - **Kategoria** (sarja/assosiatiivinen)
  - **Toistuvat operaatiot tehokkaita**
  - (Mitätöityminen, muut pienemmät erot)

Tampereen yliopisto

# Säiliön valinta tehokkuuden perusteella

| Säiliö | Yleinen lisäys/poisto | Erik. lisäys alku/loppu | Erik. poisto alku/loppu | Haku (järjestysnro) | Haku (arvo/avain) | Suurin/pienin yms. |
|---|---|---|---|---|---|---|
| vector | O(n) | - / O(1)* | - / O(1)* | O(1) | ( O(n) ) | ( O(n) ) |
| deque | O(n) | O(1) / O(1) | O(1) / O(1) | O(1) | ( O(n) ) | ( O(n) ) |
| list | O(1) | O(1) / O(1) | O(1) / O(1) | ( O(n) ) | ( O(n) ) | ( O(n) ) |
| (array) | - | - | - | O(1) | ( O(n) ) | ( O(n) ) |
| (forward_list) | O(1) | O(1) / ( O(n) ) | O(1) / ( O(n) ) | O(n) | ( O(n) ) | ( O(n) ) |
| unordered_ map/set | O(n) ≈ Θ(1) | - | - | - | O(n) ≈ Θ(1) | ( O(n) ) |
| map/set | O(log n) | - | - | ( O(n) ) | O(log n) | O(1) |
| (stack) | - | - / O(1) | - / O(1) | - | - | - |
| (queue) | - | O(1) / - | - / O(1) | - | - | - |
| (priority_queue) | O(log n) | - | - | - | - | O(1) |

Tampereen yliopisto

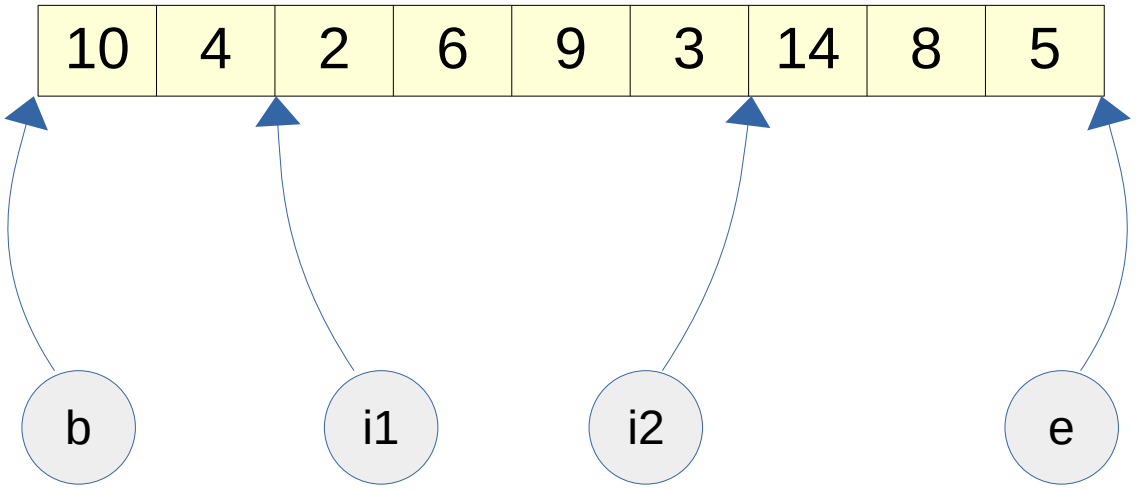
# STL:n iteraattorit

*COMP.CS.300 Tietorakenteet ja algoritmit 1*

*Matti Rintala (matti.rintala@tuni.fi)*

# Iteraattoreiden idea

- Kirjanmerkki säiliöön
- Säiliön läpikäynti
- Osaväli: 2 iteraattoria (C++20 myös: ranges)



# Iteraattoreiden rooli STL:ssä

- Säiliöt
  - `begin()`, `end()`
  - Alkion lisäys tiettyyn paikkaan
  - Alkion poisto (tai välin)
  - Operaation tuloksena paikka (tai väli)
- Algoritmit
  - Paikan *ja säiliön* ilmaiseminen
  - Toimintavälin ilmaiseminen
  - Operaation tuloksena paikka (tai väli)
- *Käänteisiteraattorit* `rbegin()`, `rend()`

Tampereen yliopisto

# Iteraattoreiden tehokkuus, iteraattorikategoriat

*COMP.CS.300 Tietorakenteet ja algoritmit 1*

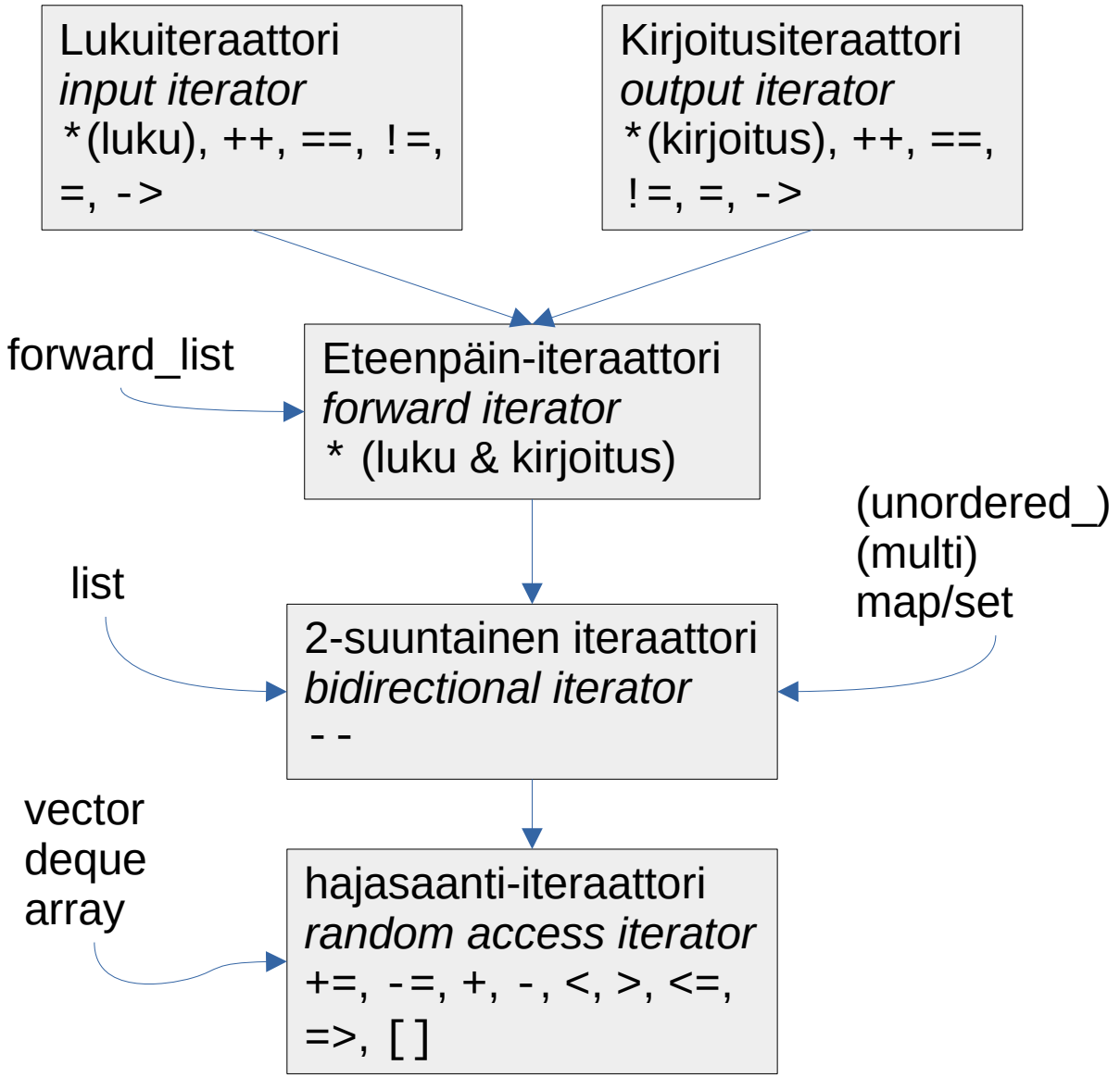
*Matti Rintala (matti.rintala@tuni.fi)*

# Iteraattoreiden tehokkuus & kategoriat

- Iteraattoreiden operaatiot (++, *, ...) *vakioaikaisia* O(1)
- Säiliöstä riippuu, mitä operaatioita iteraattori tukee
- Iteraattorikategoriat: mitä operaatioita tuetaan
- Algoritmit voivat vaatia tietyn kategorian iteraattoreita

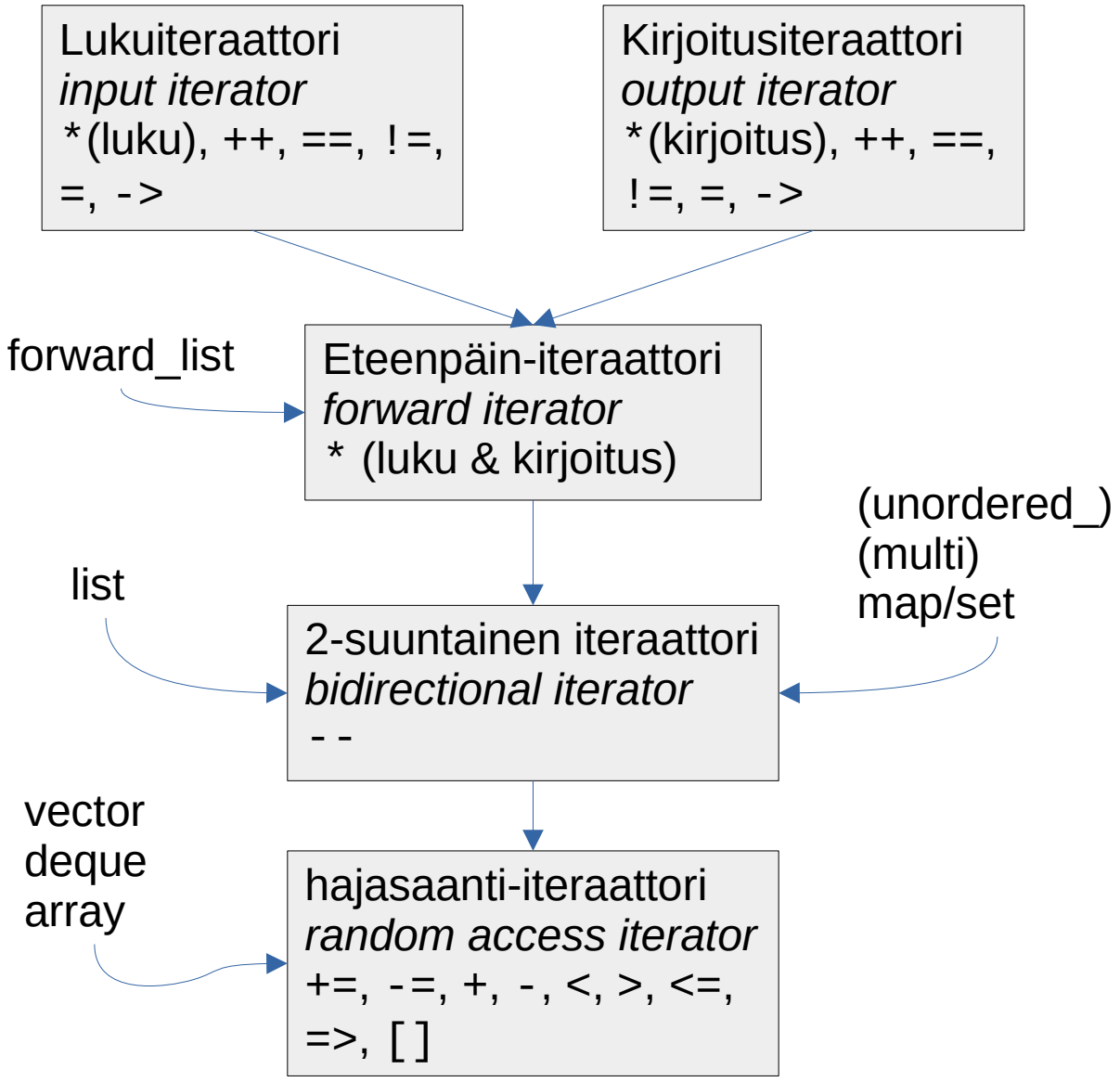
Tampereen yliopisto

# Iteraattoreiden tehokkuus & kategoriat



# Iteraattoreiden tehokkuus & kategoriat

- Iteraattoreiden operaatiot (++, *, ...) *vakioaikaisia* O(1)
- Säiliöstä riippuu, mitä operaatioita iteraattori tukee
- Iteraattorikategoriat: mitä operaatioita tuetaan
- Algoritmit voivat vaatia tietyn kategorian iteraattoreita



Tampereen yliopisto

# Säiliöviittausten mitätöityminen (invalidation)

*COMP.CS.300 Tietorakenteet ja algoritmit 1*

*Matti Rintala (matti.rintala@tuni.fi)*

Tampereen yliopisto

# Mitätöityminen (invalidation)

- Mitätöitynyt iteraattori ei enää viittaa (oikeaan) paikkaan säiliössä lisäyksen/poiston jälkeen
- Mitätöitynyttä iteraattoria *ei saa* käyttää (uuden arvon saa sijoittaa)
- Käytön seuraus määrittelemätön (kaatuminen/sekoaminen/???)

# Mitätöityminen (invalidation)

```
vector<int> v={ 1,2,3,4};
auto i = v.begin();
auto j = i+1; // i:tä seuraava
v.erase(i);
*j = 3; // !!! j mitätöitynyt!
```

# Mitätöityminen ja säiliön valinta

- Eri säiliöillä eri säännöt mitätöitymiseen
- Toinen valintakriteeri tehokkuuden lisäksi (usein kompromissi)
- vector ja deque: säännöt monimutkaiset
- unordered_map/set: turvallinen poiston suhteen, lisäys mitätöi
- map/set ja (forward_)list lähes turvallisia

Tampereen yliopisto

# Mitätöityminen ja säiliön valinta

| Säiliö | Lisäyksen jälkeen | Poiston jälkeen | Huom. |
|---|---|---|---|
| vector | Mitätön! | - | *Kapasiteetti* muuttui |
| | Ok | Ok * | *Ennen* muutospaikkaa |
| | Mitätön! | Mitätön! | Muutospaikan *jälkeen* |
| deque | Mitätön! | Ok * | *1./viim.* lisäys/poisto |
| | Mitätön! | Mitätön! | *muun* lisäys/poisto |
| (forward_)list | Ok | Ok * | |
| (multi)map/set | Ok | Ok * | |
| unordered_(multi) map/set | Mitätön! | - | *Uudelleenhajautus* tapahtui |
| | Ok | Ok * | |

# Miten huomata mitätöityminen

- Huolellinen suunnittelu!!!
- Joissakin kääntäjissä STL-debug-tiloja

  Gcc: -D_GLIBCXX_DEBUG
  -D_GLIBCXX_DEBUG_PEDANTIC
- Ohjelma kaatuu: debuggeri kertoo missä?
- Ohjelma sekoaa: debuggeri/tulostukset

Tampereen yliopisto

# Mitätöityminen, osoittimet ja indeksit

- Mikä tahansa paikan ilmaisin voi mitätöityä!
- Osoitin alkioon: alkio siirtyy muistissa
- Indeksi alkioon: alkiota ennen lisää/pois alkioita
- cppreference.com:ssa täydellisempi taulukko