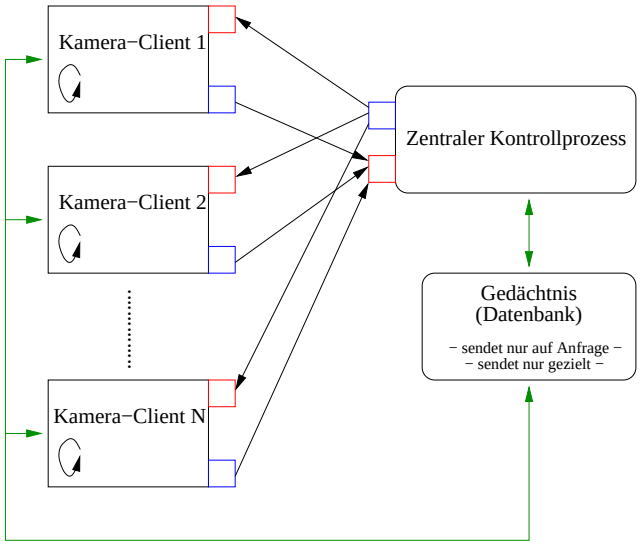
Entwurf: Kommunikationsstruktur Multi-Kamera-Netzwerk

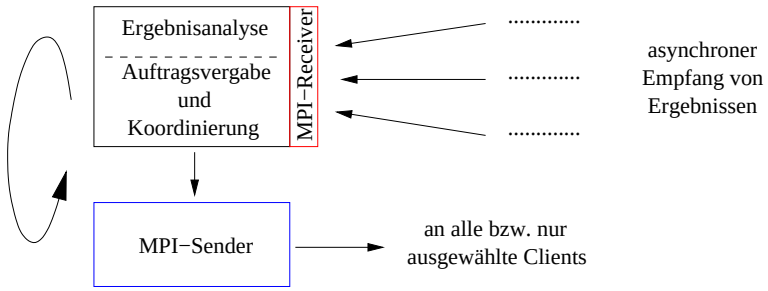
- **Legende:**
  - rote Kästchen in den Grafiken markieren nebenläufige Empfangsprozesse
  - blaue Kästchen markieren einmalige Sendefunktionen
  - grüne Pfeile kennzeichnen einen einmaligen Datenaustausch mit expliziter Anfrage
- **Globale Struktur:**



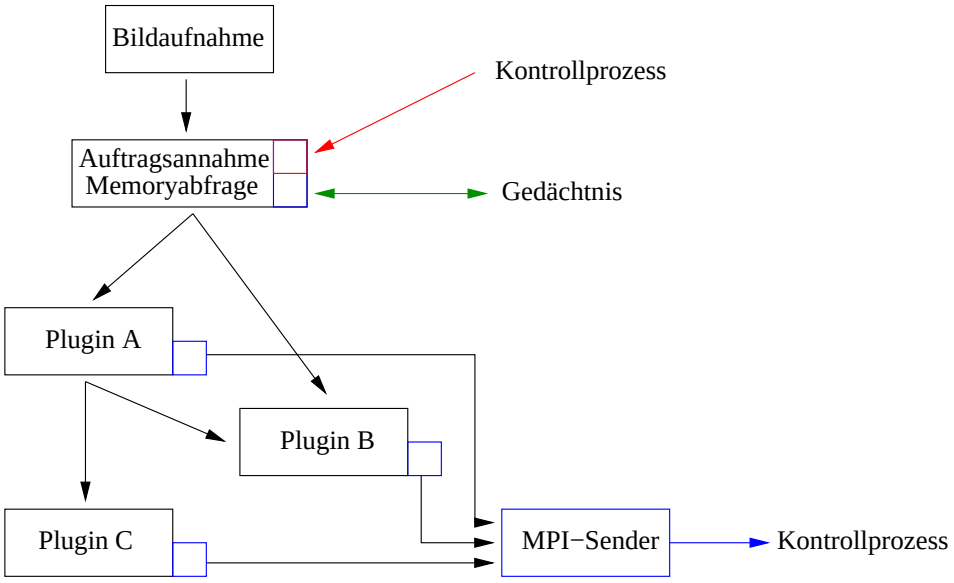
  - Bestandteile:
    a) zentraler Kontrollprozess
    läuft auf einem beliebigen Rechner und steuert/koordiniert das Netzwerk; kümmert sich um die Aufgabenverteilung, d.h. er versendet "Aufträge" an die Kamera-Clients und sammelt die Ergebnisse wieder ein
    b) Kamera-Clients
    für jede Kamera läuft ein Client auf dem zugehörigen Rechner, der sich um die Bildanalyse und den Datenaustausch kümmert
    c) Systemgedächtnis
    zentraler Prozess, der nur einmal irgendwo läuft und quasi eine Datenbank verwaltet; der Kontrollprozess und die Kamera-Clients können Anfragen an das Gedächtnis stellen und Daten eintragen/abrufen

- **Kontrollfluss:**
  - **Zentraler Kontrollprozess**
    - läuft in einer Schleife:
      1. Datenfusion und -analyse,
         Generierung von Aufträgen für die Kamera-Clients
      2. Versenden von Aufträgen
    - parallel dazu werden die asynchron versandten Ergebnisse der Kamera-Clients eingesammelt; die Clients versenden dabei auch nicht angeforderte Ereignismitteilungen, die neue Aktivitäten und Entscheidungen des Netzwerks erfordern

* Aufträge können entweder simultan an alle Kamera-Clients versendet werden, oder nur selektiv an einzelne, je nach aktueller Situation, Systemzustand und Aufgabenstellung
* der "MPI-Empfänger" des zentralen Kontrollprozesses soll als Thread parallel zur Hauptschleife laufen; er wird immer nur zu Beginn einer Schleifeniteration abgefragt (holen aller bzw. der schon empfangenen Ergebnisse), muss aber immer empfangsbereit sein, da die Kamera-Clients asynchron senden



– **Kamera-Clients:**
  * jeder Kamera-Client setzt sich aus einer Reihe von Plugins zusammen, die in eine nebenläufige Umgebung eingebettet sind ("Icewing") und nacheinander auf die Daten angewendet werden
  * sie laufen in einer Schleife, die jeweils durch das Bildaufnahme-Plugin initiiert wird
  * nach der Bildaufnahme wird ein MPI-Empfangsplugin abgefragt, das zuvor (parallel) Aufträge vom zentralen Prozess entgegen genommen hat
  * optional wird danach eine einmalige, direkte Anfrage an das Systemgedächtnis gestellt, falls Daten zur Ausführung der Aufträge noch fehlen
  * dann werden die Plugins datengetrieben abgearbeitet, wobei sie Ergebnisse produzieren, die asynchron an den zentralen Prozess gesendet werden sollen; dies kann direkt geschehen, oder durch expliziten Aufruf eines MPI-Sender-Plugins
  * die Clients senden sowohl angefragte Ergebnisse wie auch Zusatzinformationen über neue Beobachtungen



- **Systemgedächtnis:**
  - zentraler Prozess, der als Schnittstelle zu einer Datenbank fungiert
  - der Prozess selbst verhält sich passiv, d.h. er reagiert nur auf Anfragen und sendet dann bei Bedarf Informationen an den anfragenden Kamera-Client oder den zentralen Kontrollprozess

Was wir dafür bräuchten...

- am besten eine (oder mehrere?) C++-Bibliotheken, die leicht in Form von MPI-Kommunikationsobjekten in unsere Programmstrukturen eingebunden werden können und die MPI-Funktionen maskieren (z.B. Gedächtnisabfrageobjekt, Client-Kommunikationsobjekt)
- folgende Schnittstellen sollten aus Sicht der Anwendung in etwa gegeben sein:
  (Achtung: sicher noch unvollständig!)
  - für den zentralen Kontrollprozess zur Kommunikation mit den Clients und der Datenbank
    - `init_mpi_communication( )`
    - `finish_mpi_communication( )`
    - `send_message_to_all_clients( message )`
    - `send_message_to_client( message, id )`
    - `give_me_all_results( )`
    - `give_me_result_from_client( id )`
    - `tell_me_about_new_events( )`
    - `give_me_info_from_database( id )`
    - `put_info_into_database( info )`
  - für die Kamera-Clients
    - `init_mpi_communication( )`
    - `finish_mpi_communication( )`
    - `send_message_to_controlProcess( message )`
    - `get_messages_from_controlProcess( )`
    - `give_me_info_from_database( id )`
    - `put_info_into_database( info )`
  - für das Systemgedächtnis
    - ...brauchen wir eigentlich keine Extra-Funktionen, aber es muss natürlich einen MPI-Prozess geben, der Anfragen entgegen nimmt und beantwortet bzw. die Datenbank verwaltet
- ausserdem brauchen wir vermutlich ein paar allgemeine Funktionen zur Abfrage des Systemzustands, etc.:
  - `count_number_of_clients( )`
  - `is_client_alive( id )`
  - `...`
- die Bibliothek selbst muss sich um folgendes kümmern:

  - nebenläufiger Empfang und Versenden von Nachrichten
  - korrekte Adressierung und asynchroner Datenaustausch

- Daten, die verschickt werden müssen, und für die wir geeignete Strukturen brauchen:
  - Koordinaten von Punkten, Winkel, Entfernungen
  - Objekteigenschaften (Farbe, Form, Polygonzüge, etc.)
  - Nachrichten ("unbekannte Person detektiert"), geeignet codiert
  - gelegentlich komplette Bilder oder Ausschnitte
  - ...

  $\Rightarrow$ das wird sich über die Zeit sicher öfter mal ändern...